GNU Assembler

CMPE230 - Spring’24

Gökçe Uludoğan

GNU Assembler

- the assembler developed by the GNU Project.
- used to assemble the GNU operating system and the Linux kernel.
- uses AT&T assembly syntax.

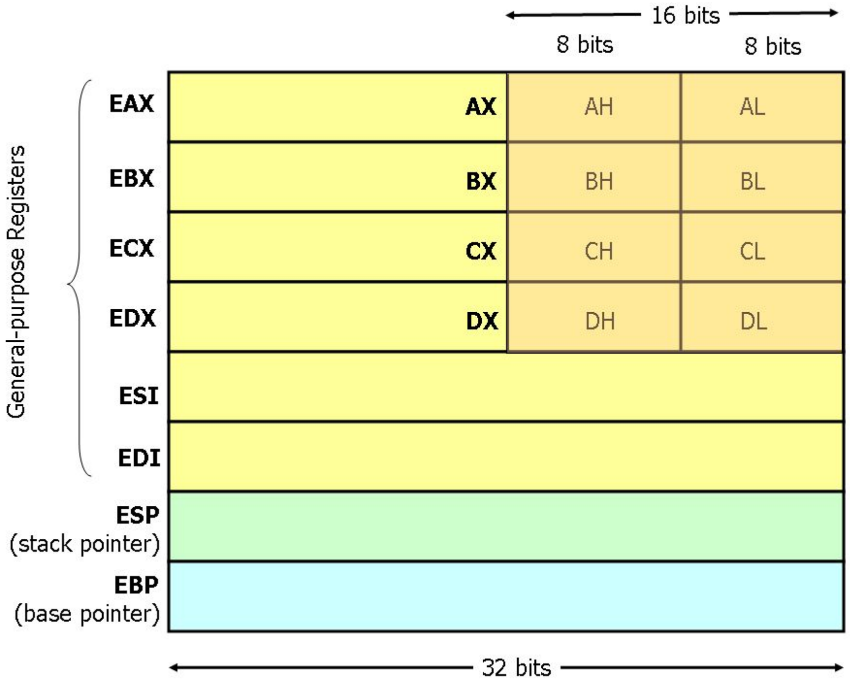
AT&T Syntax

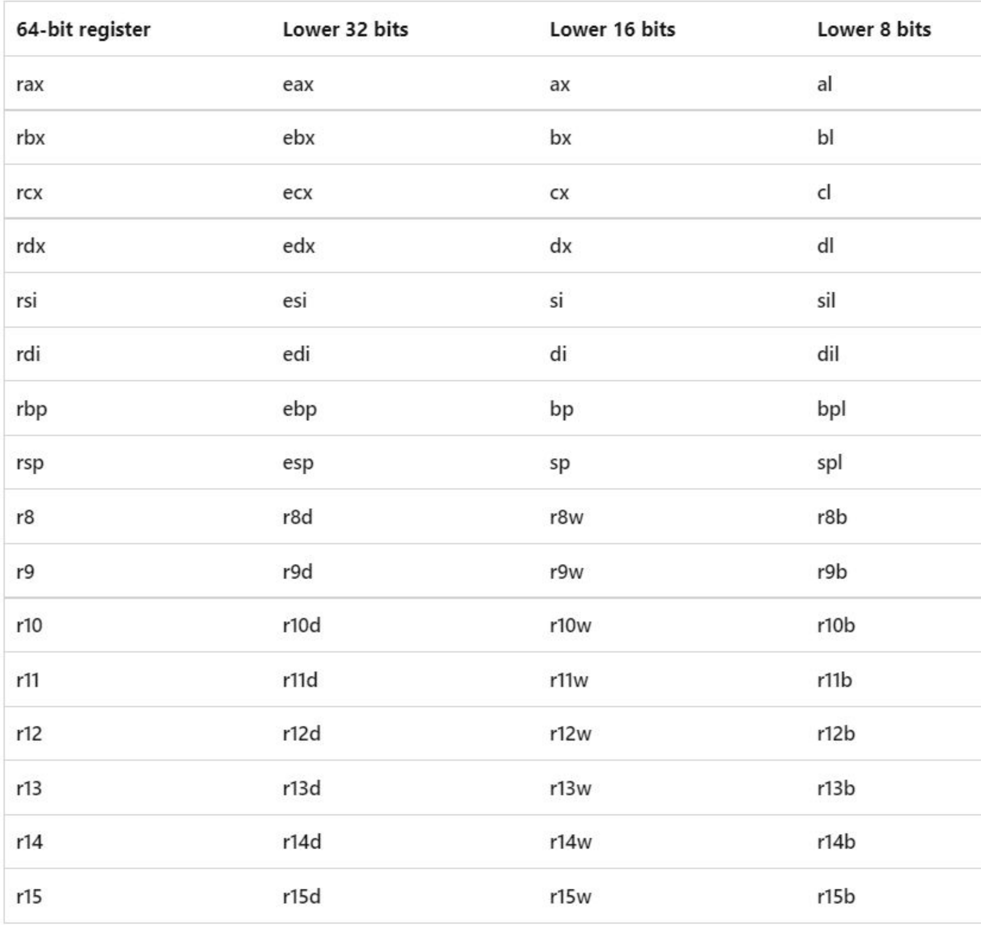
- Similar to any other assembler syntax.
- Consists of a series of directives, labels, instructions
- Composed of a mnemonic followed by a maximum of three operands
    - **the ordering of the operands are reversed.**

Intel Syntax (e.g. A86)	mnemonic destination, source
AT&T Syntax	mnemonic source, destination

https://csiflabs.cs.ucdavis.edu/~ssdavis/50/att-syntax.htm

x86 vs x64 registers





64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

x86 registers

General registers

EAX EBX ECX EDX

Segment registers

CS DS ES FS GS SS

Index and pointers

ESI EDI EBP EIP ESP

x86 registers

General registers

EAX EBX ECX EDX

Segment registers

CS DS ES FS GS SS

Index and pointers

ESI EDI EBP EIP ESP

General purpose registers

32 bits:	EAX EBX ECX EDX
16 bits:	AX BX CX DX
8 bits:	AH AL BH BL CH CL DH DL

x86 registers

General registers

EAX EBX ECX EDX

Segment registers

CS DS ES FS GS SS

Index and pointers

ESI EDI EBP EIP ESP

Segment registers hold the segment address of various items

CS : Holds the Code segment in which your program runs.

DS : Holds the Data segment that your program accesses

ES,FS,GS: These are extra segment registers available for far pointer addressing like video memory and such.

SS : Holds the Stack segment your program uses.

x86 registers

General registers

EAX EBX ECX EDX

Segment registers

CS DS ES FS GS SS

Index and pointers

ESI EDI EBP EIP ESP

Indexes and pointer and the offset part of and address.

EDI: Destination index register
Used for string, memory array copying and setting

ESI: Source index register
Used for string and memory array copying

EBP: Base pointer register
Also called frame pointer

ESP: Stack pointer register
Holds the top address of the stack

EIP: Index Pointer
Holds the offset of the next instruction

AT&T Syntax: Prefixes

- All register names must be prefixed by a '%'
    - `mov %ax, %bx`

- All literal values must be prefixed by a '$'.
    - `mov $100, %bx`
    - `mov $A, %al`
    - *`Invalid: mov %bx, $100`*

https://csiflabs.cs.ucdavis.edu/~ssdavis/50/att-syntax.htm

AT&T Syntax: Memory Addressing

- Memory is referenced in the following way:
  - offset(base, index, scale)

which is equivalent to [base + index * scale + offset] in Intel syntax.

AT&T Syntax	Intel Syntax
`100`	`[100]`
`(%eax)`	`[eax]`
`(%eax,%ebx)`	`[eax+ebx]`
`(%ecx,%ebx,2)`	`[ecx+ebx*2]`
`-100(%eax)`	`[eax-100]`

Example instructions:

- `mov  %ax,  100`
- `mov %eax, -100(%eax)`

https://csiflabs.cs.ucdavis.edu/~ssdavis/50/att-syntax.htm

AT&T Syntax: Operand Sizes

- By adding a suffix - b/w/l - to the instruction.
  - b: byte (8 bits)
  - w: word (16 bits)
  - l: long (32 bits)

- Examples
  - `movl   $100, %ebx`
  - `pushl  %eax`
  - `popw   %ax`

https://csiflabs.cs.ucdavis.edu/~ssdavis/50/att-syntax.htm

Calling Functions

- Entering a function
  - `pushl %ebp`
  - `movl %esp, %ebp`

- Returning from a function
  - `leave`

    equivalent to

  - `movl %ebp, %esp`
  - `popl %ebp`

Calling Convention

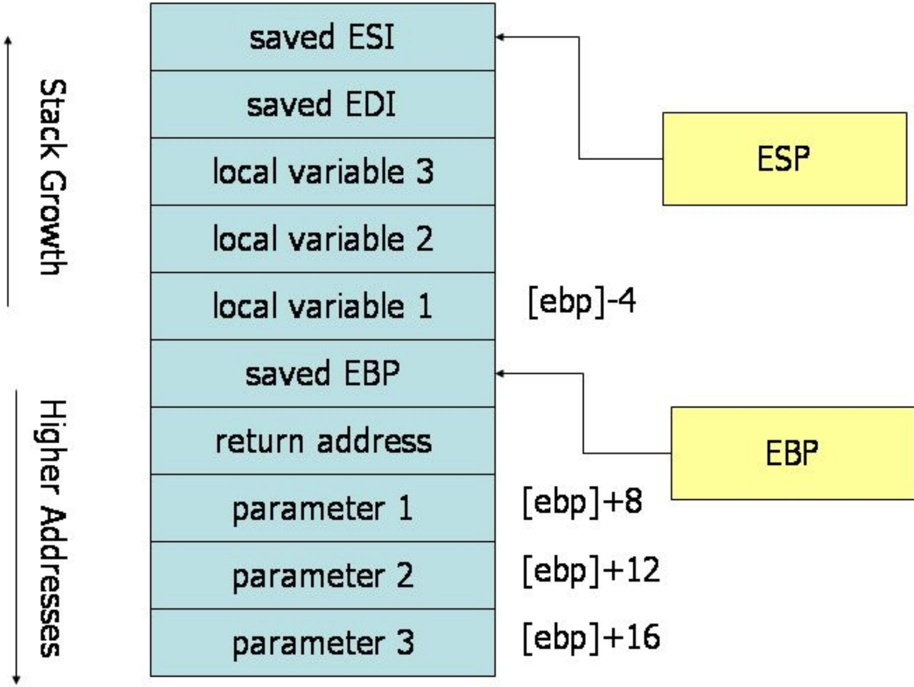
- **Caller rules:** ***Before the subroutine***

Save the contents of **caller-saved registers** that can be **modified** by the called subroutine: EAX, ECX, EDX.

To **pass parameters** to the subroutine, **push them onto the stack** before the call.

To call the subroutine, use the `call` instruction. This instruction ***places the return address*** on top of the parameters on the stack.

```
push (%ebx)     /* Push last parameter first */
push $216       /* Push the second parameter */
push %eax       /* Push first parameter last */
call myFunc     /* Call the function */
add $12, %esp
```



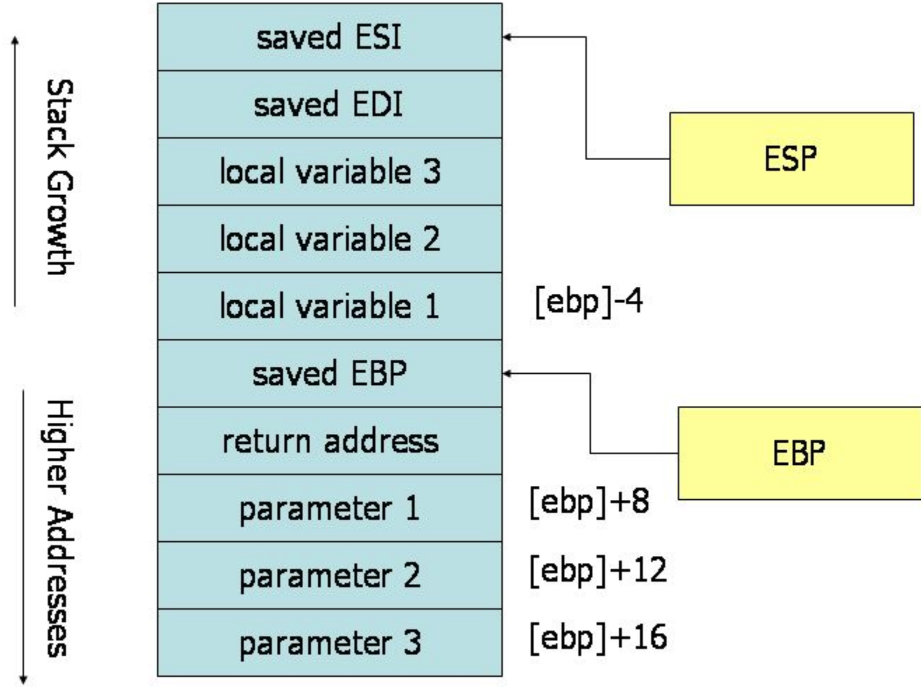
https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.htm

Calling Convention

- **Caller rules: *After the subroutine***

Remove the parameters from stack. This restores the stack to its state before the call was performed.

Restore the contents of caller-saved registers (EAX, ECX, EDX) by popping them off of the stack. The caller can assume that no other registers were modified by the subroutine.



Stack Growth

Higher Addresses

saved ESI

saved EDI

local variable 3

local variable 2

local variable 1

saved EBP

return address

parameter 1

parameter 2

parameter 3

ESP

EBP

[ebp]-4

[ebp]+8

[ebp]+12

[ebp]+16

https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.htm

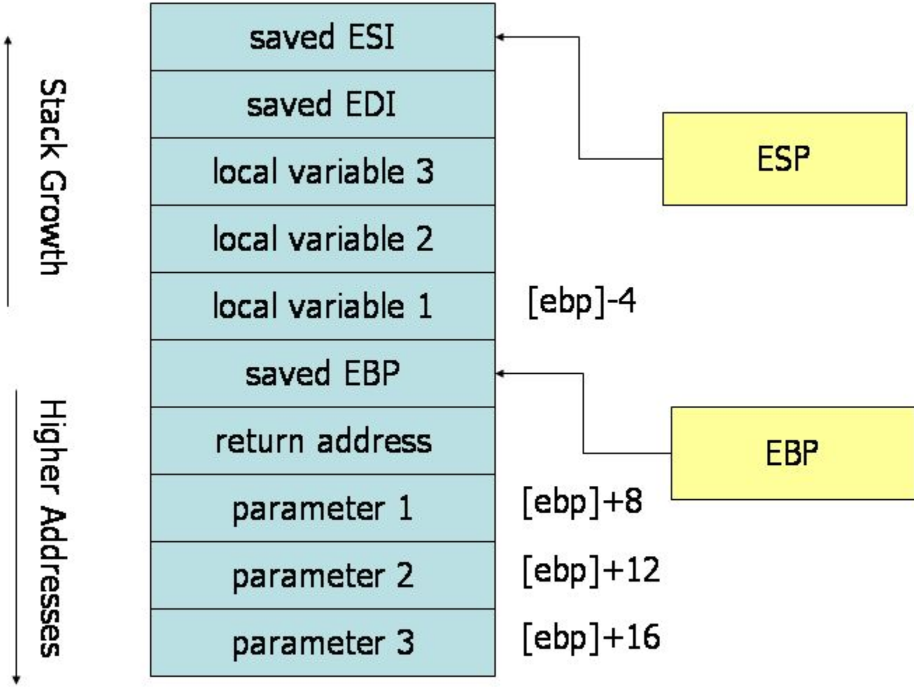
Calling Convention

- ***Callee rules:** Before the subroutine body*

  Push the value of **EBP** onto the stack, and then copy the value of **ESP into EBP**

  Allocate local variables by making space on the stack (i.e., `sub $12, %esp`)

  Save the values of the **callee-saved registers: EBX, EDI, and ESI**



https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.htm

Calling Convention

- **Callee rules: *After the subroutine body***

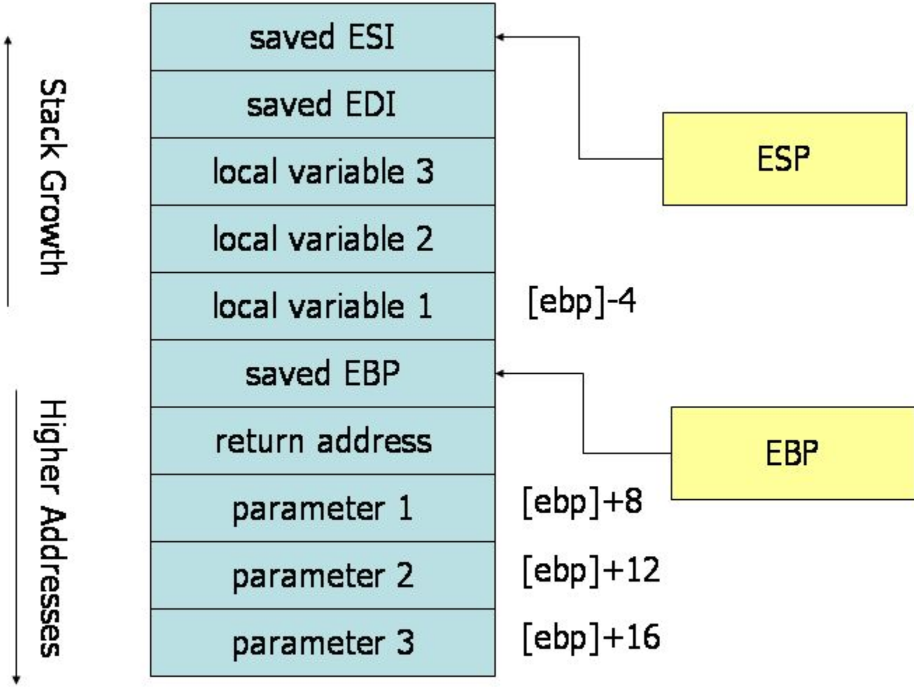
  Leave the **return value in EAX.**

  Restore the old values of any **callee-saved** registers (EDI and ESI) that were modified

  Deallocate local variables (`mov %ebp, %esp`)

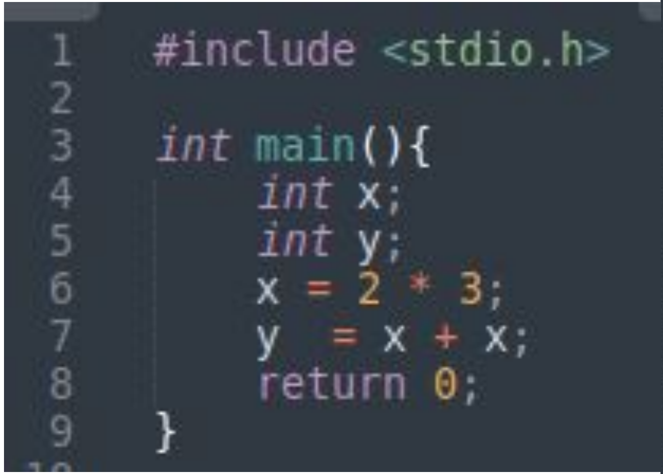
  Restore the caller's base pointer (`pop %ebp`)

  Return to the caller (`ret`)

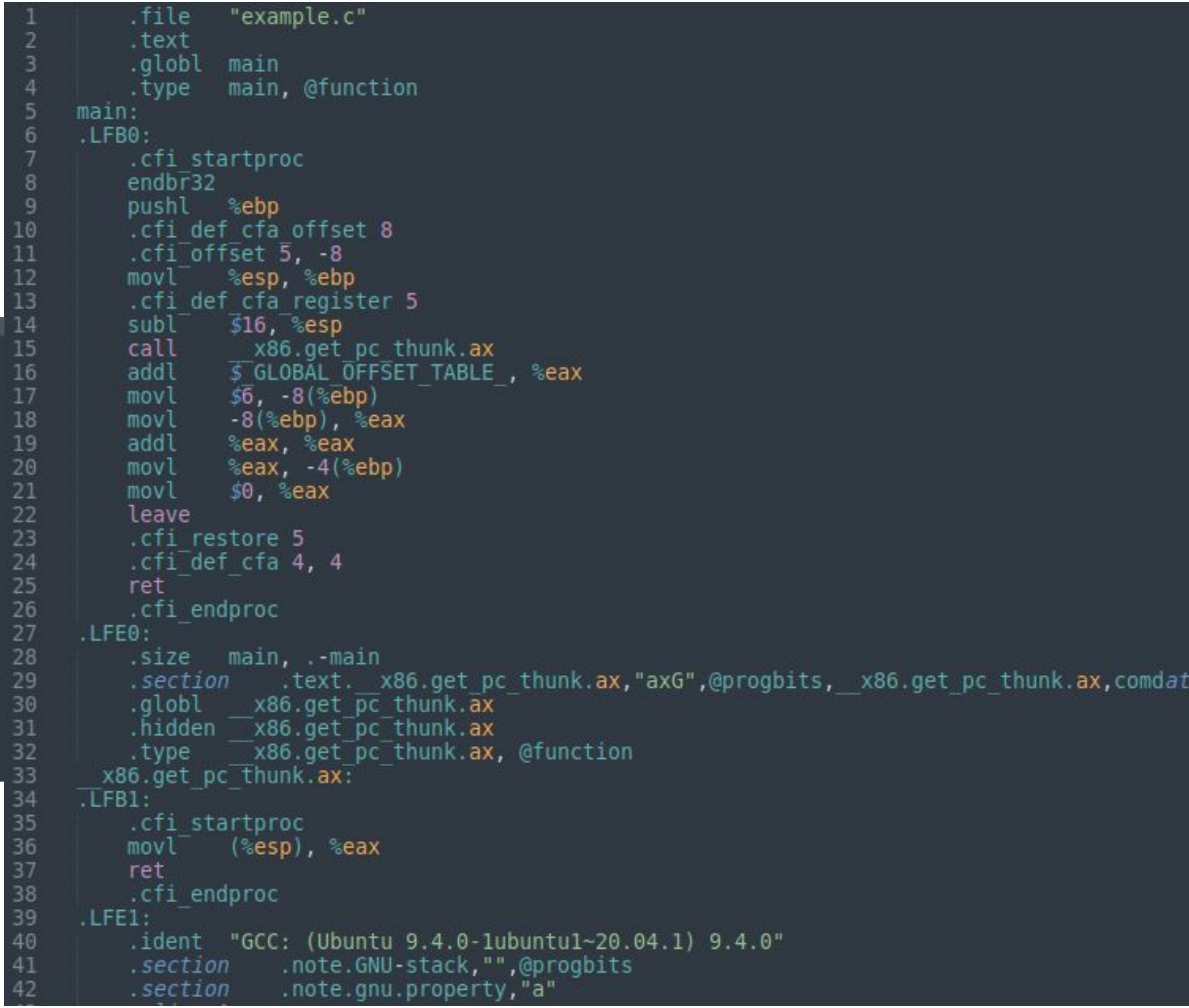


https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.htm

Example

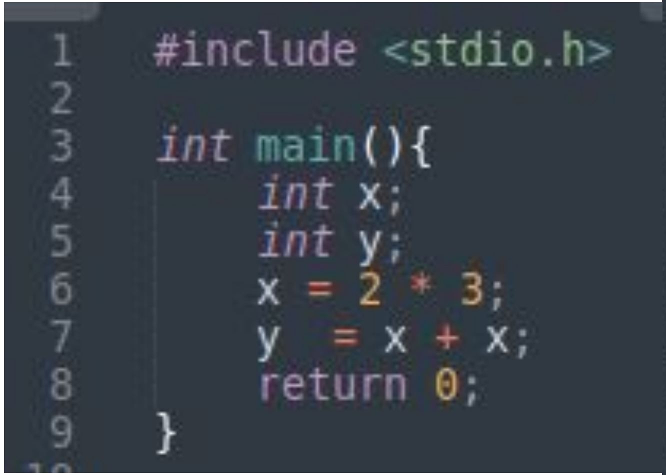


```c
#include <stdio.h>

int main(){
	int x;
	int y;
	x = 2 * 3;
	y  = x + x;
	return 0;
}
```

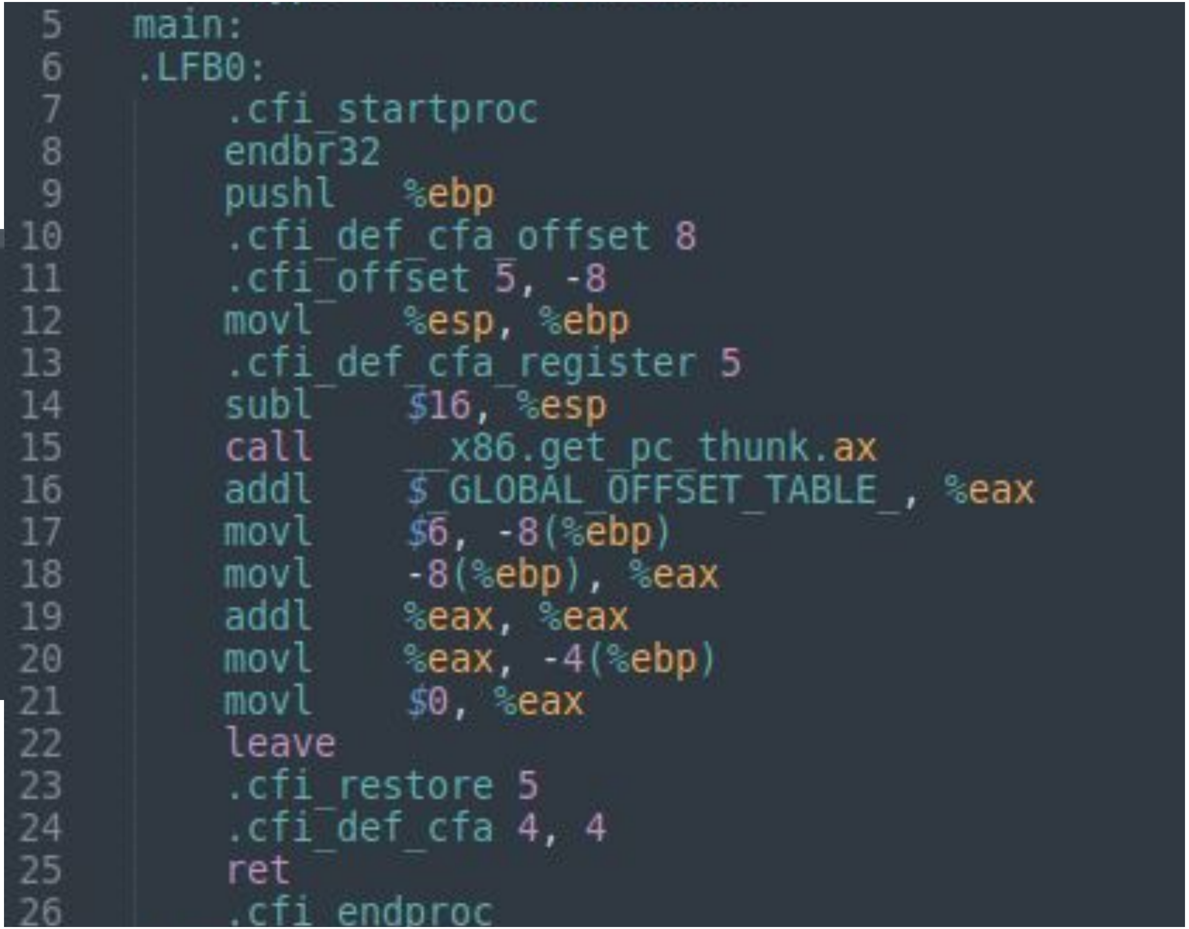


```
	.file	"example.c"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	$6, -8(%ebp)
	movl	-8(%ebp), %eax
	addl	%eax, %eax
	movl	%eax, -4(%ebp)
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.section	.text.__x86.get_pc_thunk.ax,"axG",@progbits,__x86.get_pc_thunk.ax,comdat
	.globl	__x86.get_pc_thunk.ax
	.hidden	__x86.get_pc_thunk.ax
	.type	__x86.get_pc_thunk.ax, @function
__x86.get_pc_thunk.ax:
.LFB1:
	.cfi_startproc
	movl	(%esp), %eax
	ret
	.cfi_endproc
.LFE1:
	.ident	"GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"
	.section	.note.GNU-stack,"",@progbits
	.section	.note.gnu.property,"a"
```

Example

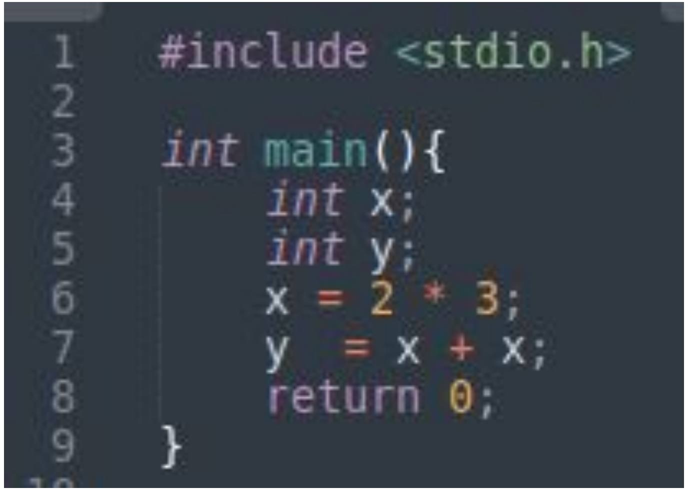


```c
#include <stdio.h>

int main(){
	int x;
	int y;
	x = 2 * 3;
	y  = x + x;
	return 0;
}
```



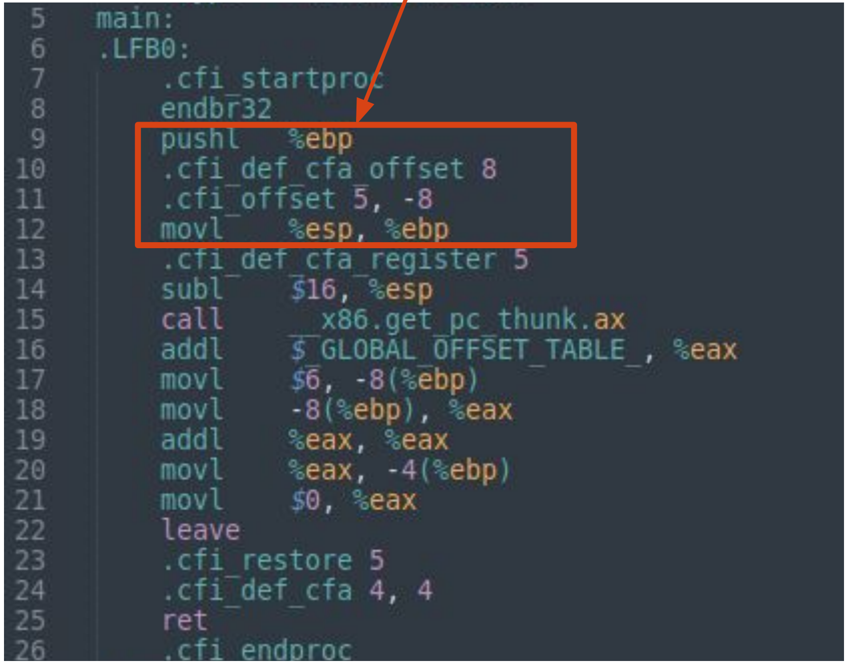
```asm
main:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	$6, -8(%ebp)
	movl	-8(%ebp), %eax
	addl	%eax, %eax
	movl	%eax, -4(%ebp)
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

Example



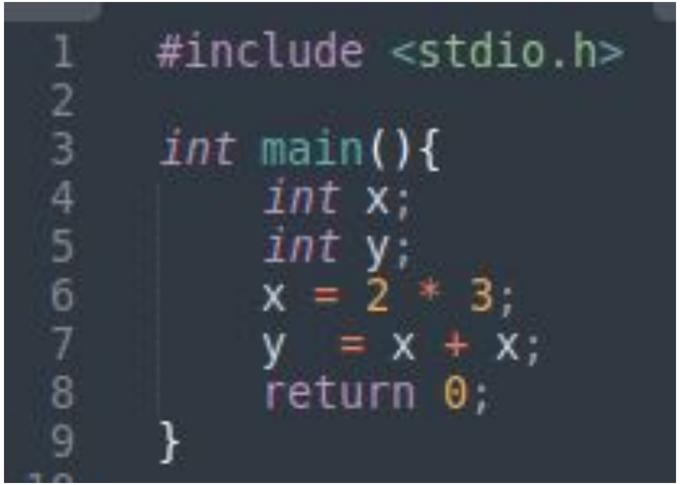
```c
#include <stdio.h>

int main(){
	int x;
	int y;
	x = 2 * 3;
	y  = x + x;
	return 0;
}
```

Entering a function



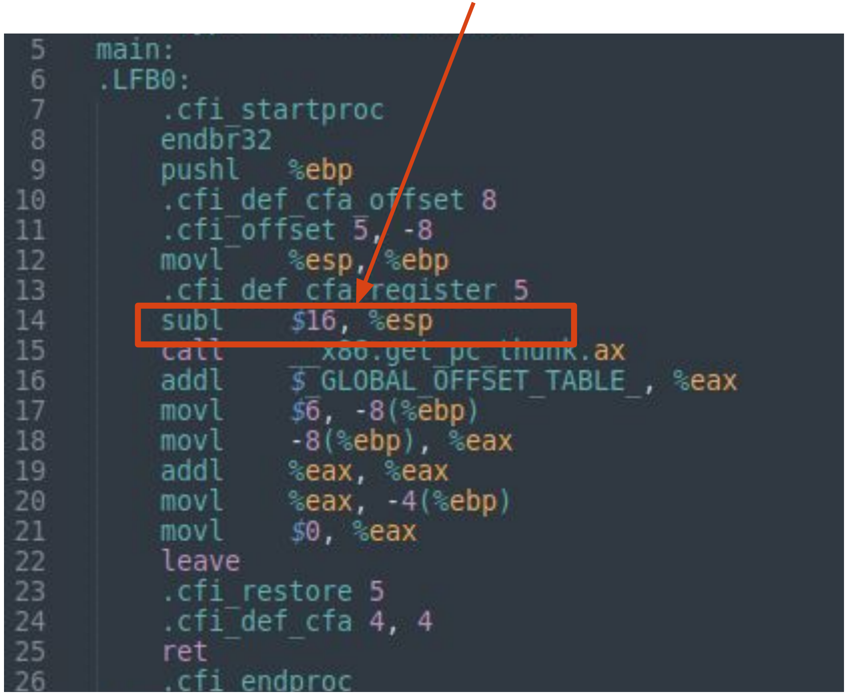
```
main:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	$6, -8(%ebp)
	movl	-8(%ebp), %eax
	addl	%eax, %eax
	movl	%eax, -4(%ebp)
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

Example



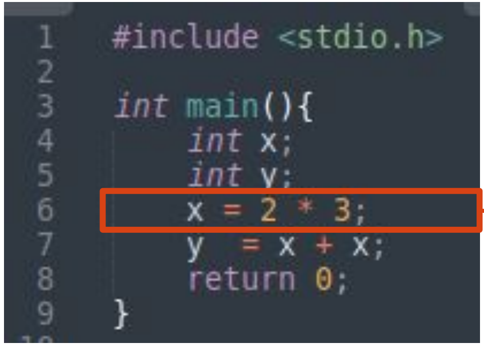
```c
#include <stdio.h>

int main(){
        int x;
        int y;
        x = 2 * 3;
        y  = x + x;
        return 0;
}
```

Moving stack pointer to allocate space

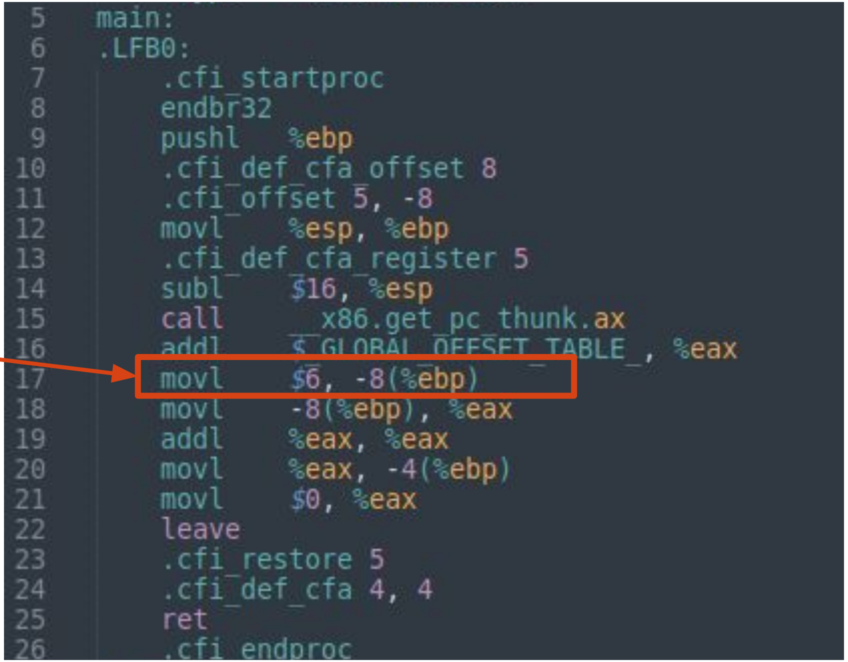


```asm
main:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    $6, -8(%ebp)
        movl    -8(%ebp), %eax
        addl    %eax, %eax
        movl    %eax, -4(%ebp)
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

Example

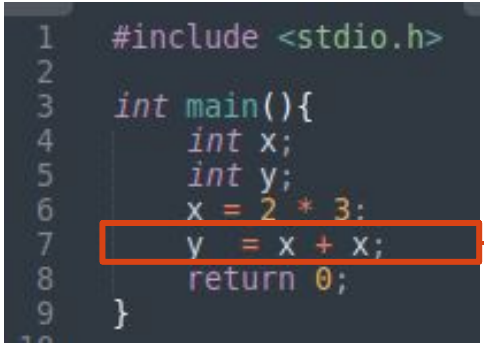


```c
#include <stdio.h>

int main(){
	int x;
	int y;
	x = 2 * 3;
	y  = x + x;
	return 0;
}
```

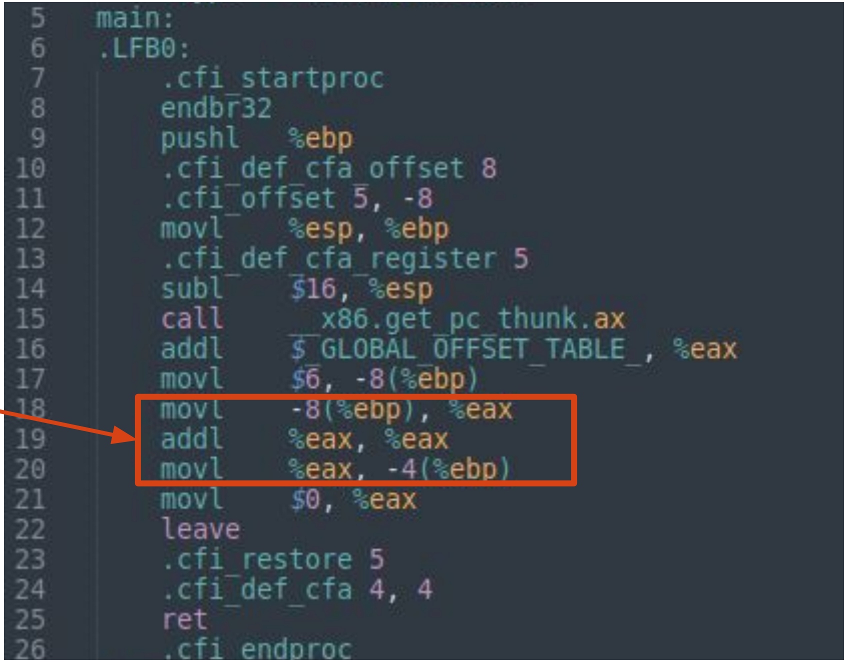


```
main:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	$6, -8(%ebp)
	movl	-8(%ebp), %eax
	addl	%eax, %eax
	movl	%eax, -4(%ebp)
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

Example

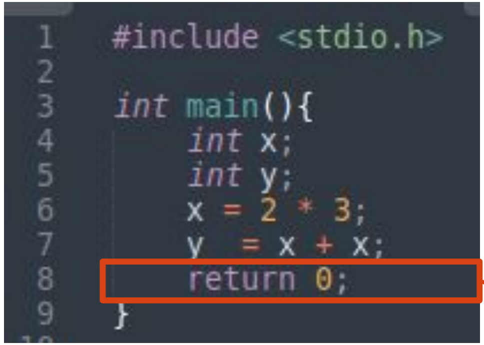


```
#include <stdio.h>

int main(){
    int x;
    int y;
    x = 2 * 3;
    y  = x + x;
    return 0;
}
```

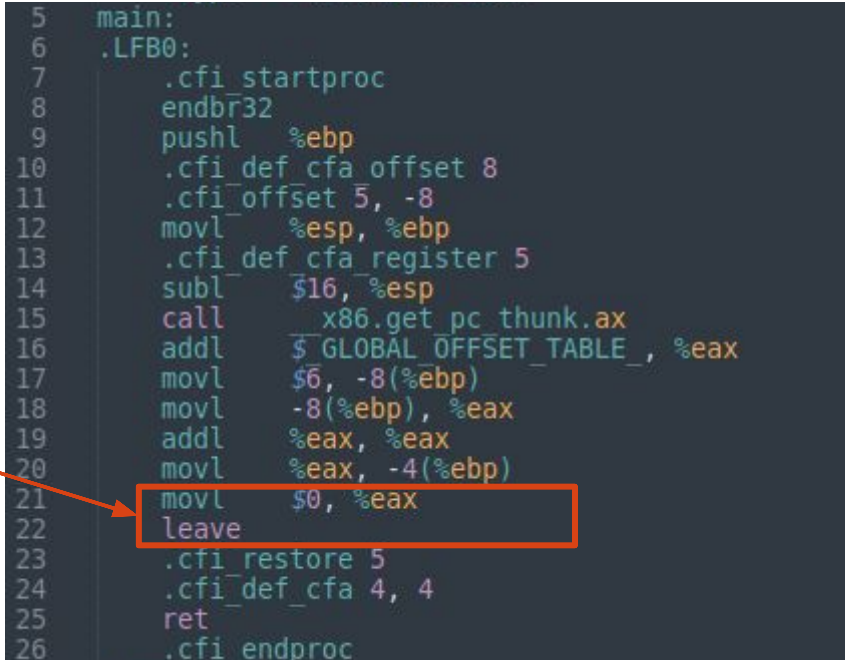


```
main:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    $6, -8(%ebp)
        movl    -8(%ebp), %eax
        addl    %eax, %eax
        movl    %eax, -4(%ebp)
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

Example

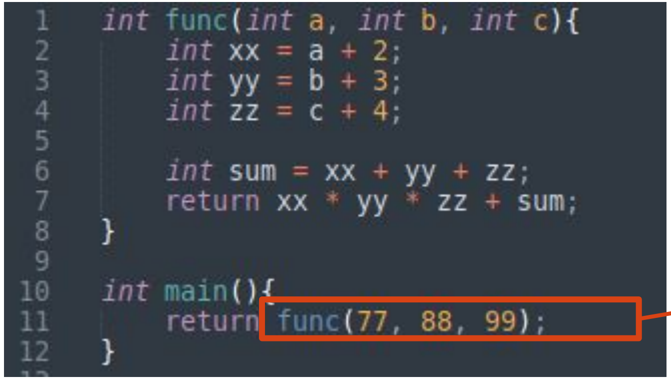


```c
#include <stdio.h>

int main(){
    int x;
    int y;
    x = 2 * 3;
    y  = x + x;
    return 0;
}
```

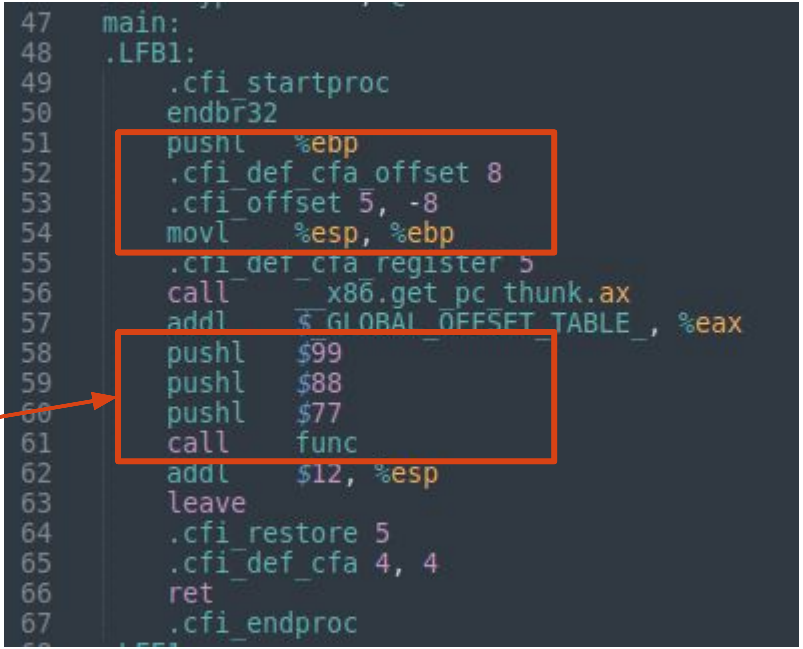


```asm
main:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    $6, -8(%ebp)
        movl    -8(%ebp), %eax
        addl    %eax, %eax
        movl    %eax, -4(%ebp)
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

Example 2

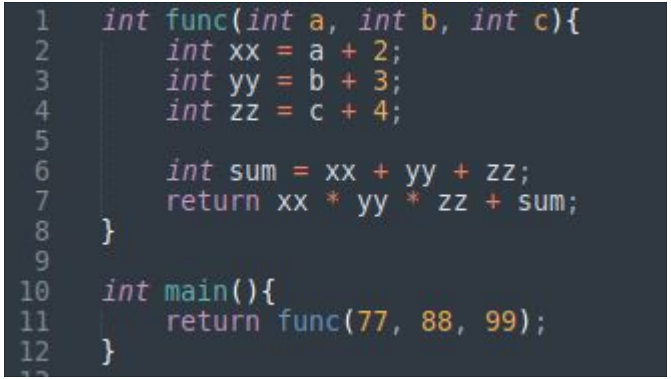


```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```

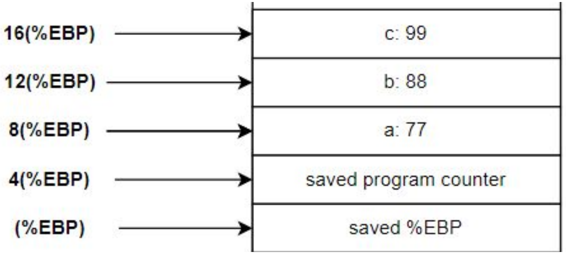


```
main:
.LFB1:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	pushl	$99
	pushl	$88
	pushl	$77
	call	func
	addl	$12, %esp
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

Example 2



```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```



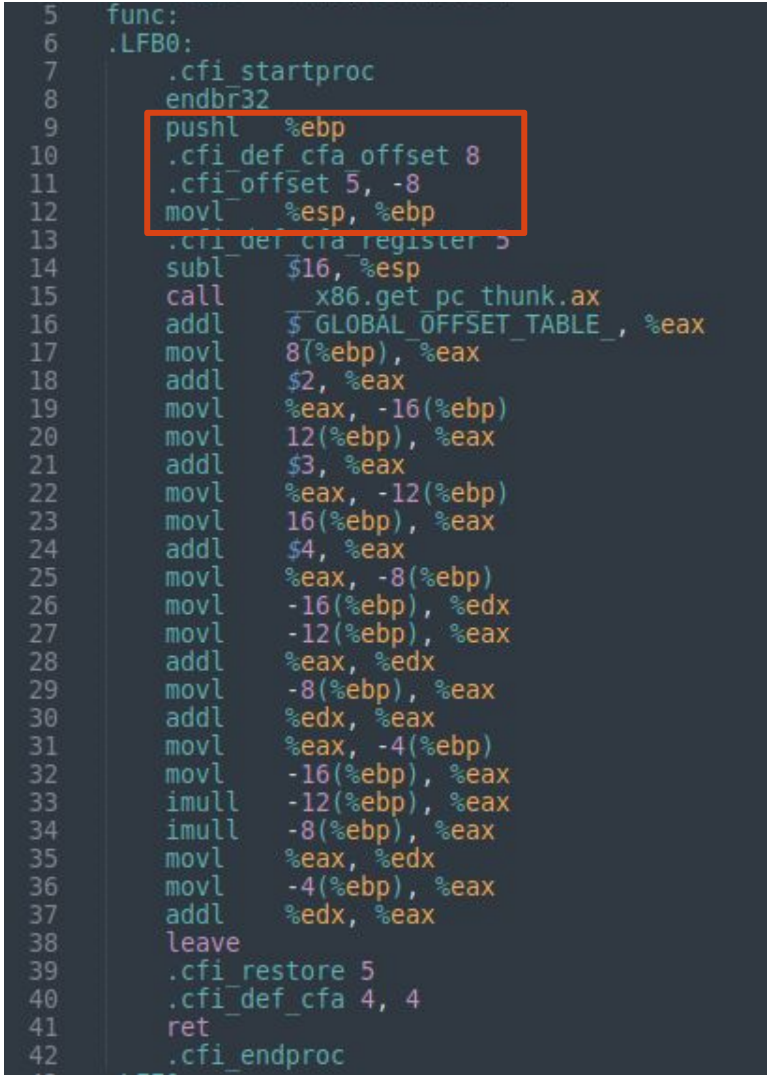
16(%EBP) → c: 99

12(%EBP) → b: 88

8(%EBP) → a: 77

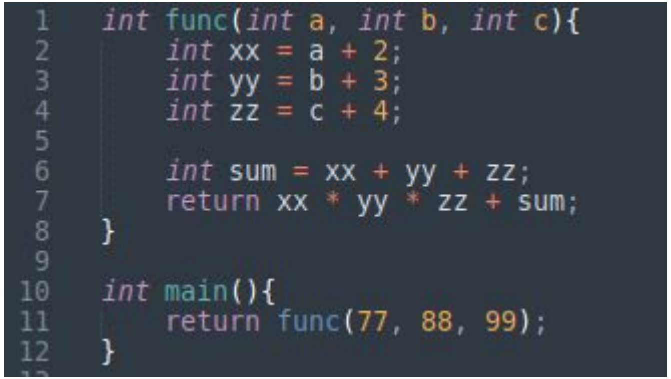
4(%EBP) → saved program counter

(%EBP) → saved %EBP

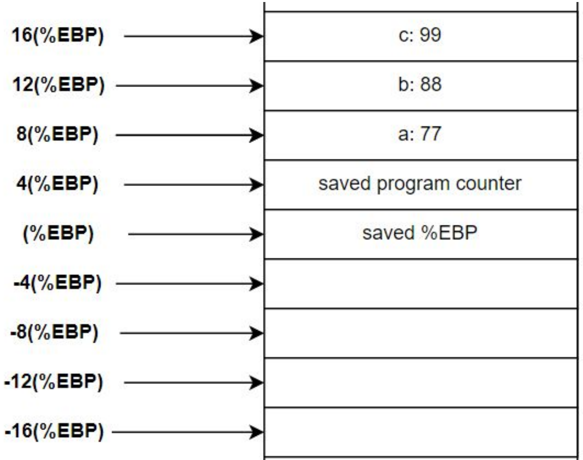


```
    func:
    .LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    8(%ebp), %eax
        addl    $2, %eax
        movl    %eax, -16(%ebp)
        movl    12(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -12(%ebp)
        movl    16(%ebp), %eax
        addl    $4, %eax
        movl    %eax, -8(%ebp)
        movl    -16(%ebp), %edx
        movl    -12(%ebp), %eax
        addl    %eax, %edx
        movl    -8(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        movl    -16(%ebp), %eax
        imull   -12(%ebp), %eax
        imull   -8(%ebp), %eax
        movl    %eax, %edx
        movl    -4(%ebp), %eax
        addl    %edx, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

Example 2



```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```



16(%EBP) → c: 99

12(%EBP) → b: 88

8(%EBP) → a: 77

4(%EBP) → saved program counter

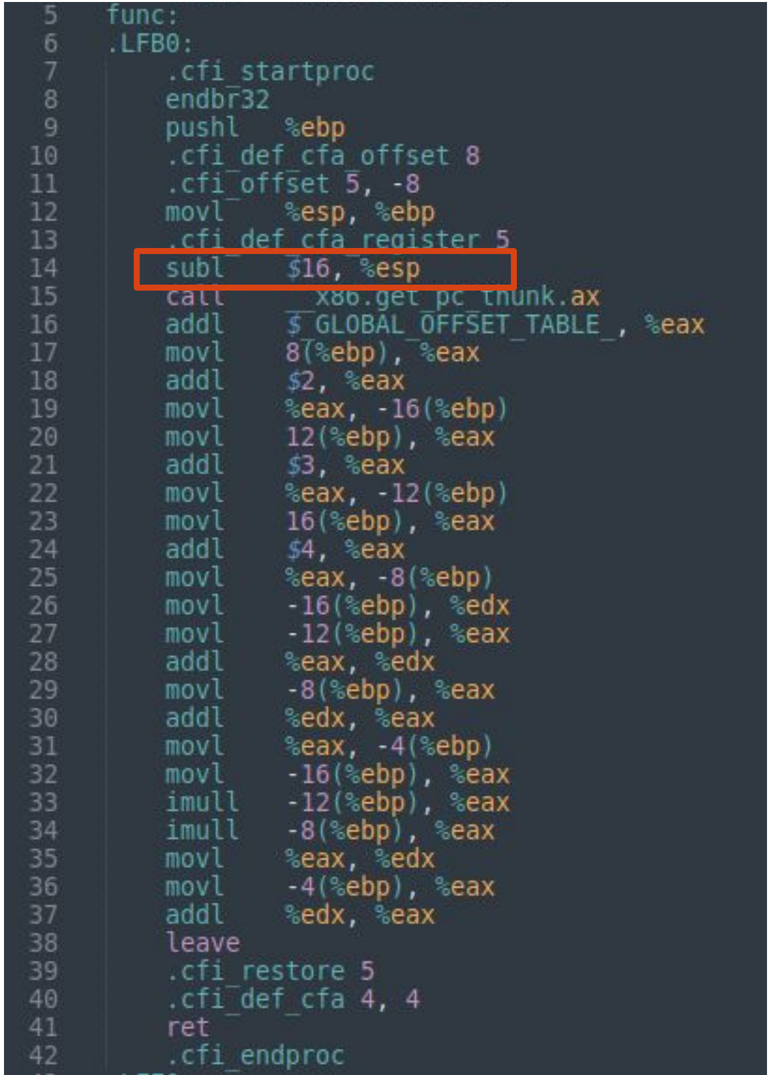
(%EBP) → saved %EBP

-4(%EBP) →

-8(%EBP) →

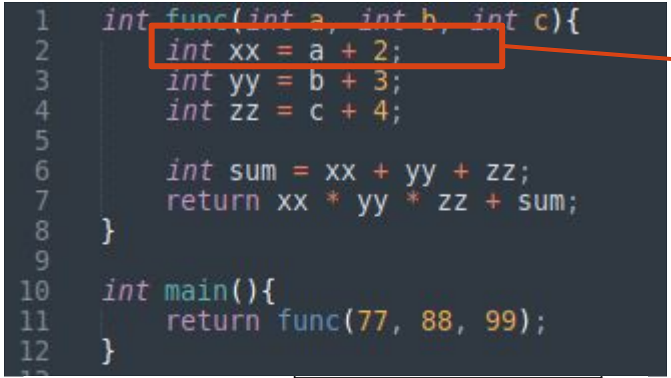
-12(%EBP) →

-16(%EBP) →

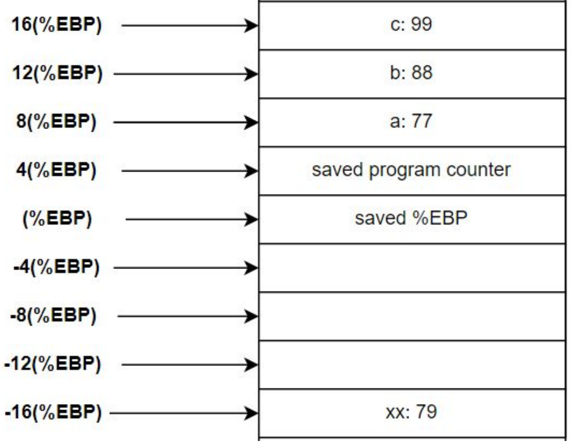


```
func:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    8(%ebp), %eax
        addl    $2, %eax
        movl    %eax, -16(%ebp)
        movl    12(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -12(%ebp)
        movl    16(%ebp), %eax
        addl    $4, %eax
        movl    %eax, -8(%ebp)
        movl    -16(%ebp), %edx
        movl    -12(%ebp), %eax
        addl    %eax, %edx
        movl    -8(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        movl    -16(%ebp), %eax
        imull   -12(%ebp), %eax
        imull   -8(%ebp), %eax
        movl    %eax, %edx
        movl    -4(%ebp), %eax
        addl    %edx, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

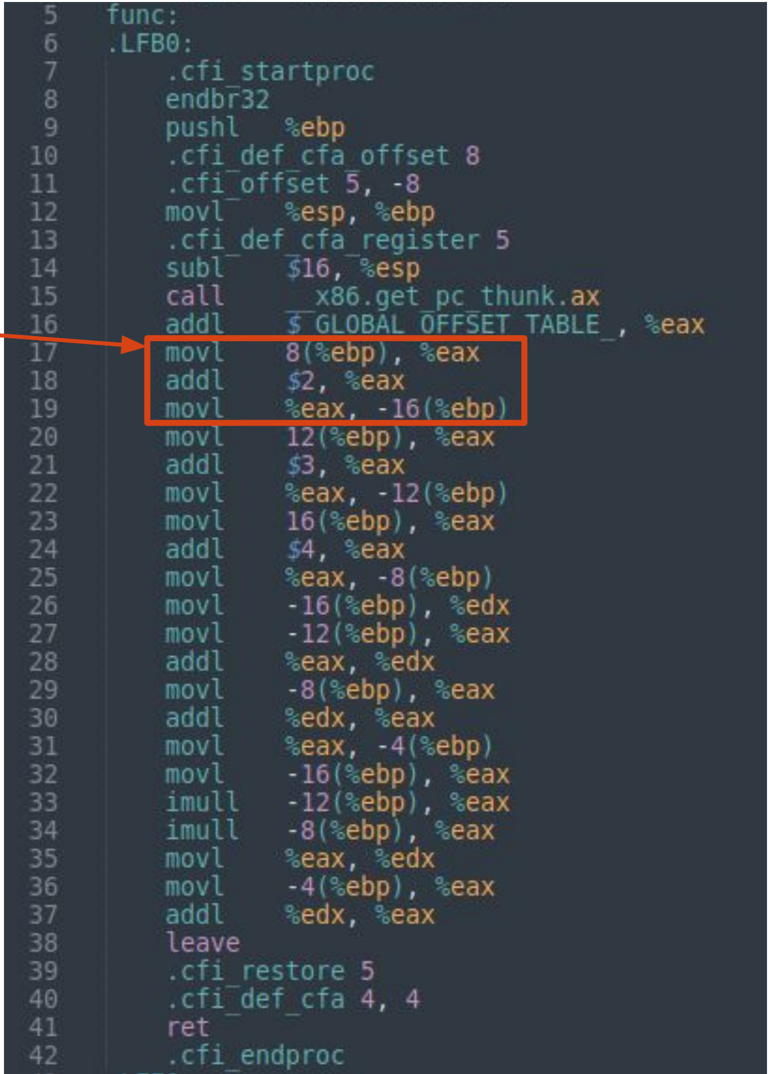
Example 2



```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```

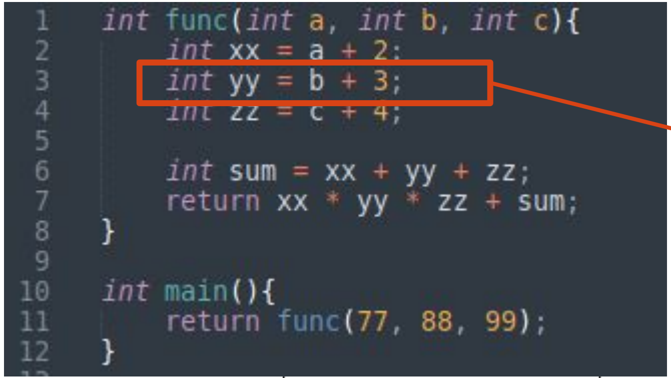


16(%EBP)	c: 99
12(%EBP)	b: 88
8(%EBP)	a: 77
4(%EBP)	saved program counter
(%EBP)	saved %EBP
-4(%EBP)	
-8(%EBP)	
-12(%EBP)	
-16(%EBP)	xx: 79

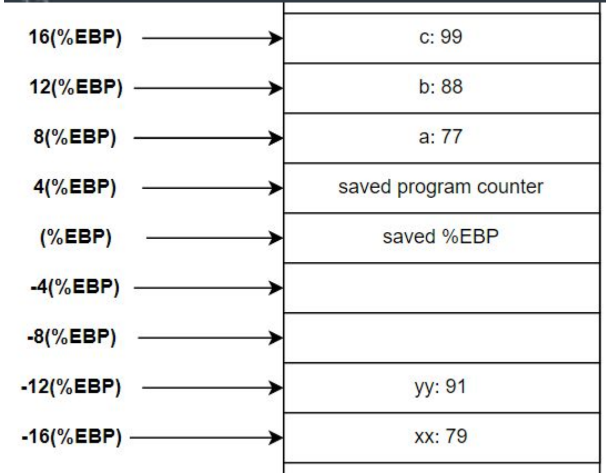


```
func:
.LFB0:
    .cfi_startproc
    endbr32
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    subl    $16, %esp
    call    __x86.get_pc_thunk.ax
    addl    $_GLOBAL_OFFSET_TABLE_, %eax
    movl    8(%ebp), %eax
    addl    $2, %eax
    movl    %eax, -16(%ebp)
    movl    12(%ebp), %eax
    addl    $3, %eax
    movl    %eax, -12(%ebp)
    movl    16(%ebp), %eax
    addl    $4, %eax
    movl    %eax, -8(%ebp)
    movl    -16(%ebp), %edx
    movl    -12(%ebp), %eax
    addl    %eax, %edx
    movl    -8(%ebp), %eax
    addl    %edx, %eax
    movl    %eax, -4(%ebp)
    movl    -16(%ebp), %eax
    imull   -12(%ebp), %eax
    imull   -8(%ebp), %eax
    movl    %eax, %edx
    movl    -4(%ebp), %eax
    addl    %edx, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
```

Example 2



```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```



16(%EBP) → c: 99

12(%EBP) → b: 88

8(%EBP) → a: 77

4(%EBP) → saved program counter

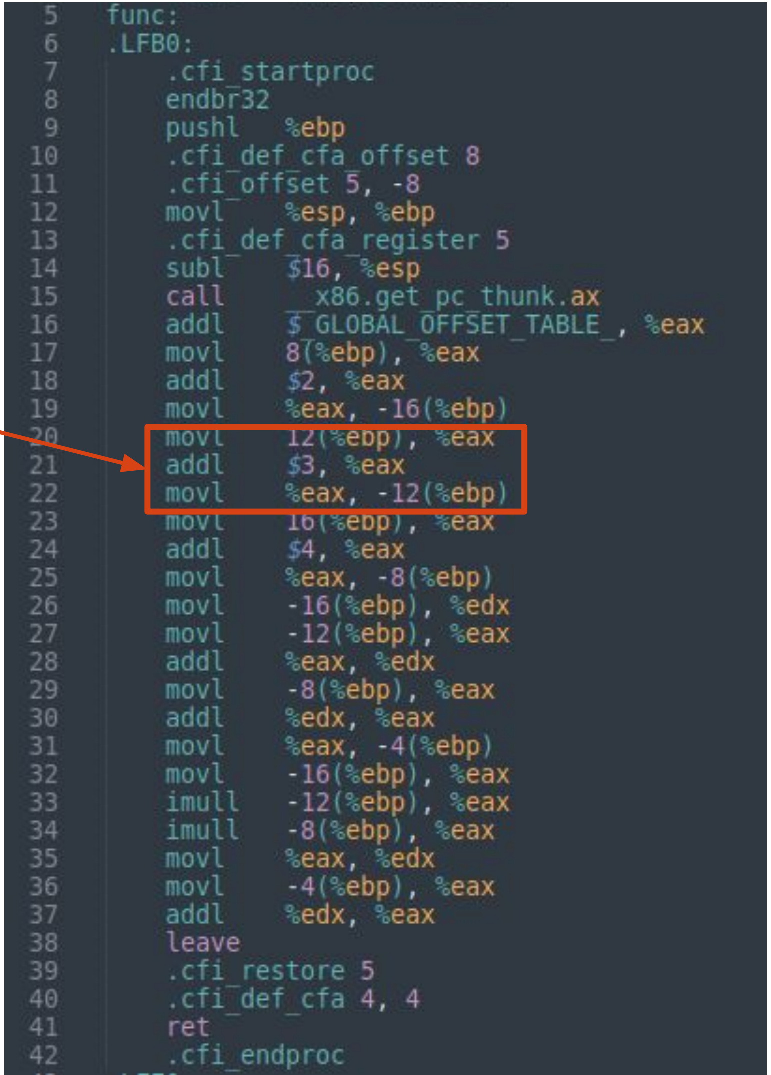
(%EBP) → saved %EBP

-4(%EBP) →

-8(%EBP) →

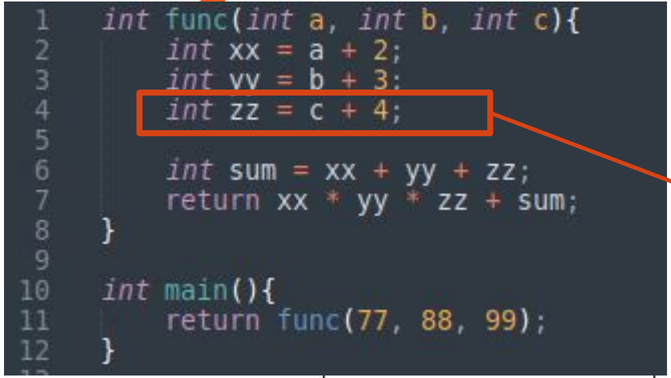
-12(%EBP) → yy: 91

-16(%EBP) → xx: 79

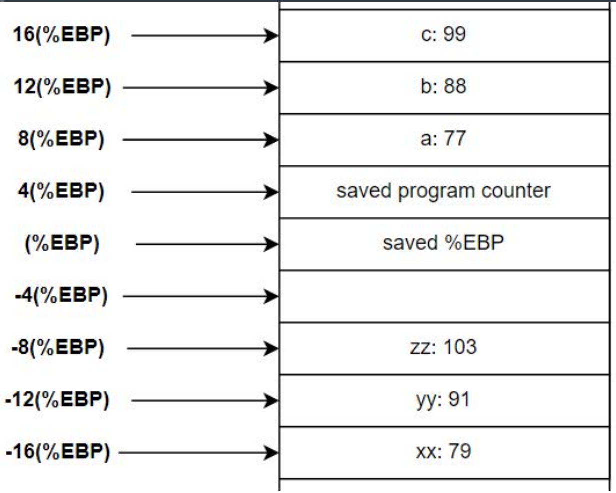


```
func:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	8(%ebp), %eax
	addl	$2, %eax
	movl	%eax, -16(%ebp)
	movl	12(%ebp), %eax
	addl	$3, %eax
	movl	%eax, -12(%ebp)
	movl	16(%ebp), %eax
	addl	$4, %eax
	movl	%eax, -8(%ebp)
	movl	-16(%ebp), %edx
	movl	-12(%ebp), %eax
	addl	%eax, %edx
	movl	-8(%ebp), %eax
	addl	%edx, %eax
	movl	%eax, -4(%ebp)
	movl	-16(%ebp), %eax
	imull	-12(%ebp), %eax
	imull	-8(%ebp), %eax
	movl	%eax, %edx
	movl	-4(%ebp), %eax
	addl	%edx, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

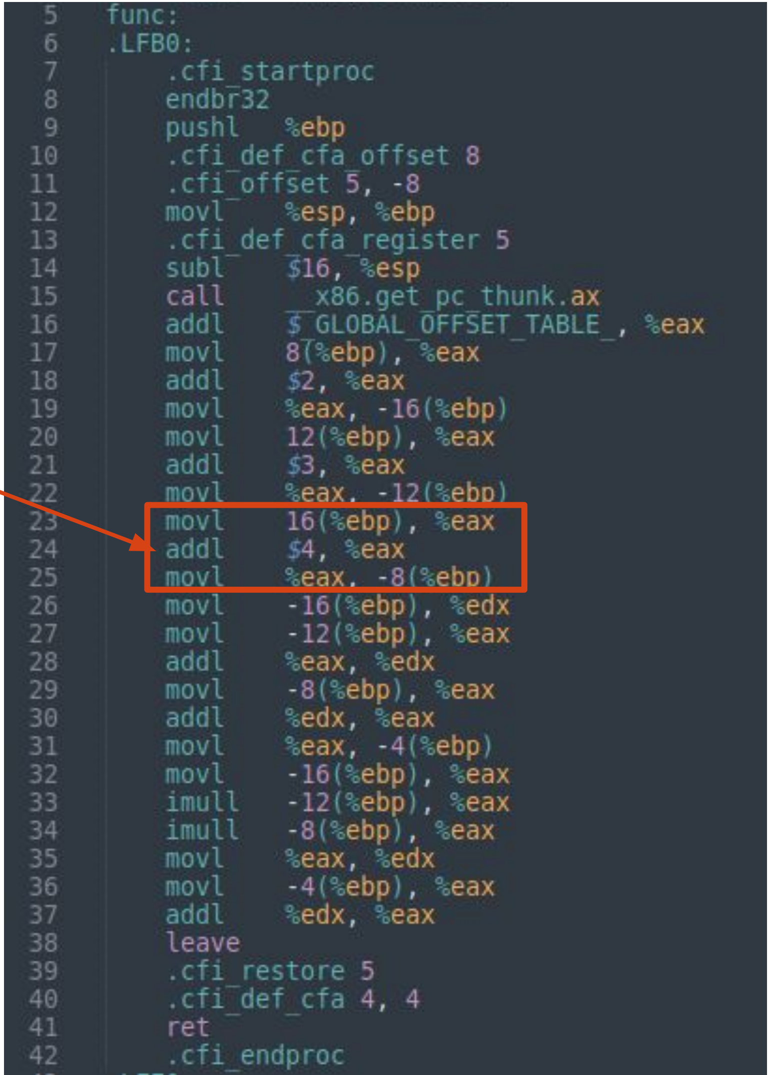
Example 2



```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```

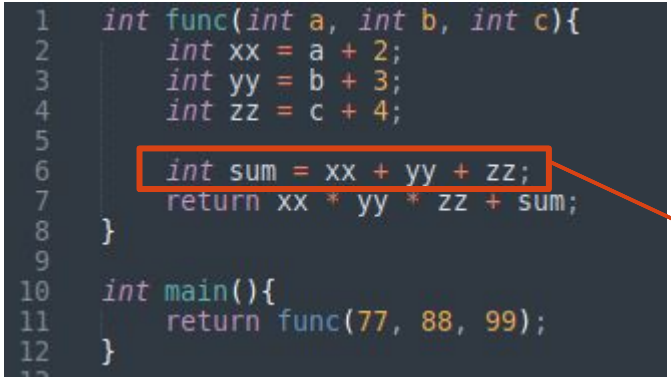


16(%EBP)	c: 99
12(%EBP)	b: 88
8(%EBP)	a: 77
4(%EBP)	saved program counter
(%EBP)	saved %EBP
-4(%EBP)	
-8(%EBP)	zz: 103
-12(%EBP)	yy: 91
-16(%EBP)	xx: 79

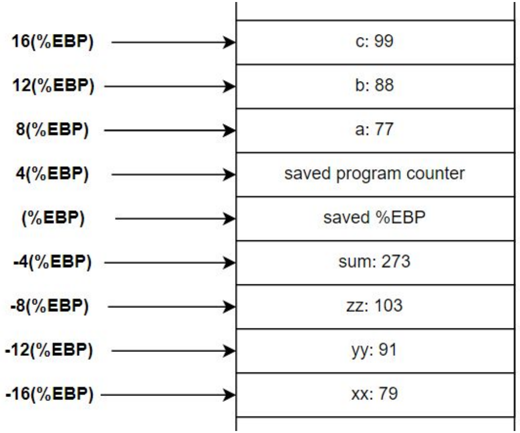


```
func:
.LFB0:
    .cfi_startproc
    endbr32
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    subl    $16, %esp
    call    __x86.get_pc_thunk.ax
    addl    $_GLOBAL_OFFSET_TABLE_, %eax
    movl    8(%ebp), %eax
    addl    $2, %eax
    movl    %eax, -16(%ebp)
    movl    12(%ebp), %eax
    addl    $3, %eax
    movl    %eax, -12(%ebp)
    movl    16(%ebp), %eax
    addl    $4, %eax
    movl    %eax, -8(%ebp)
    movl    -16(%ebp), %edx
    movl    -12(%ebp), %eax
    addl    %eax, %edx
    movl    -8(%ebp), %eax
    addl    %edx, %eax
    movl    %eax, -4(%ebp)
    movl    -16(%ebp), %eax
    imull   -12(%ebp), %eax
    imull   -8(%ebp), %eax
    movl    %eax, %edx
    movl    -4(%ebp), %eax
    addl    %edx, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
```

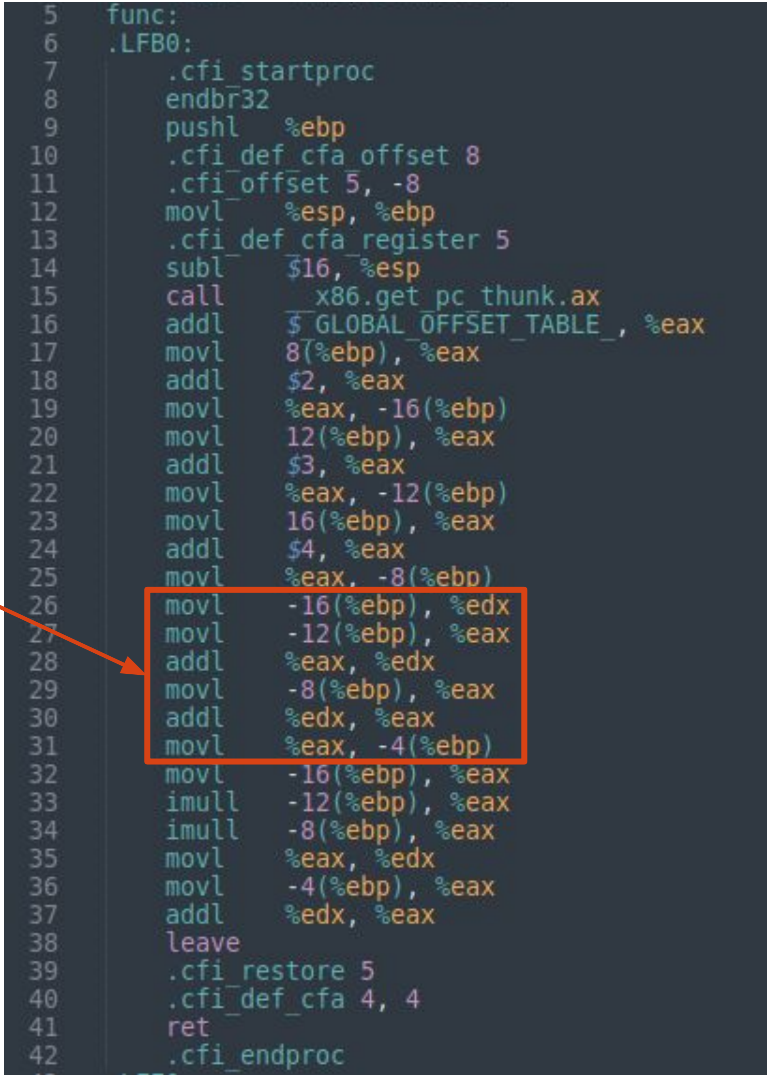
Example 2



```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```

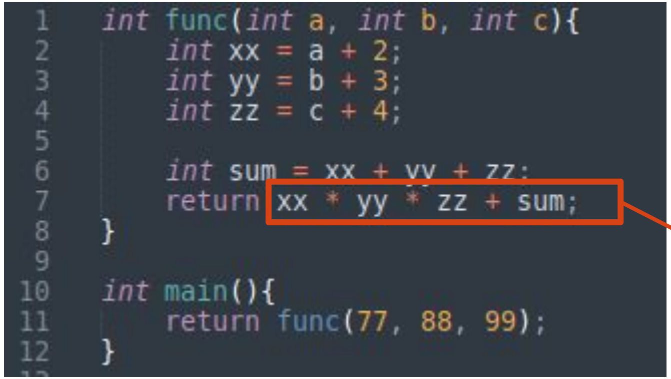


16(%EBP)	c: 99
12(%EBP)	b: 88
8(%EBP)	a: 77
4(%EBP)	saved program counter
(%EBP)	saved %EBP
-4(%EBP)	sum: 273
-8(%EBP)	zz: 103
-12(%EBP)	yy: 91
-16(%EBP)	xx: 79

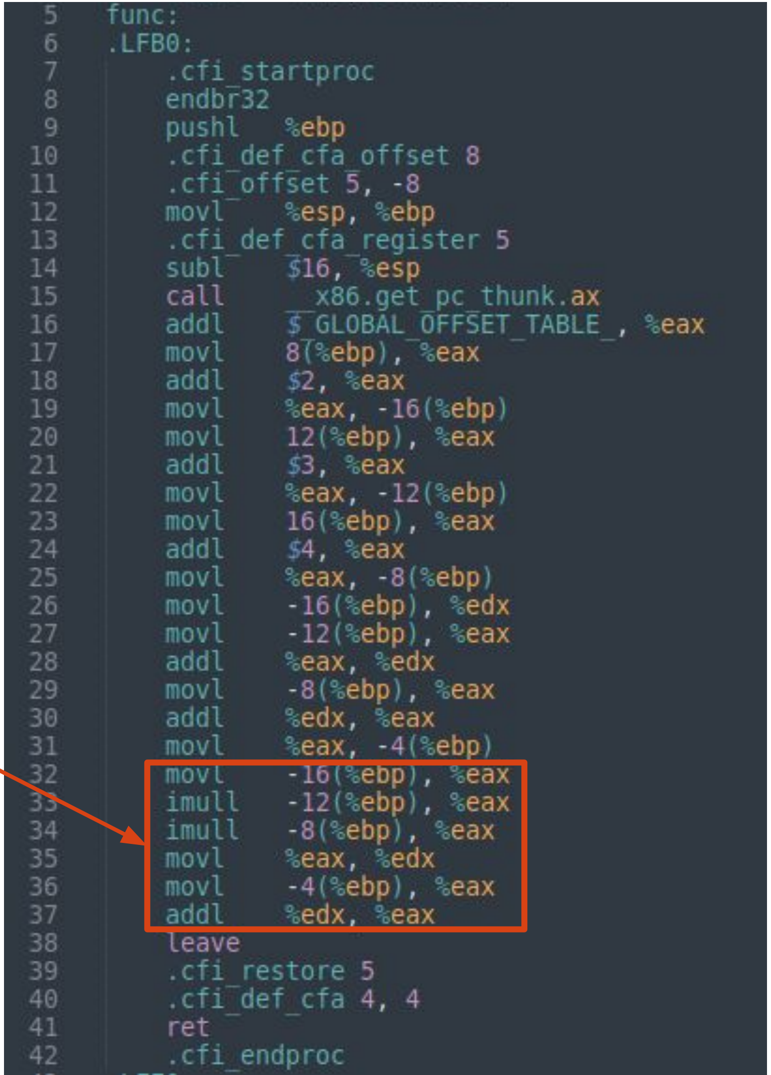


```asm
func:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	8(%ebp), %eax
	addl	$2, %eax
	movl	%eax, -16(%ebp)
	movl	12(%ebp), %eax
	addl	$3, %eax
	movl	%eax, -12(%ebp)
	movl	16(%ebp), %eax
	addl	$4, %eax
	movl	%eax, -8(%ebp)
	movl	-16(%ebp), %edx
	movl	-12(%ebp), %eax
	addl	%eax, %edx
	movl	-8(%ebp), %eax
	addl	%edx, %eax
	movl	%eax, -4(%ebp)
	movl	-16(%ebp), %eax
	imull	-12(%ebp), %eax
	imull	-8(%ebp), %eax
	movl	%eax, %edx
	movl	-4(%ebp), %eax
	addl	%edx, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

Example 2



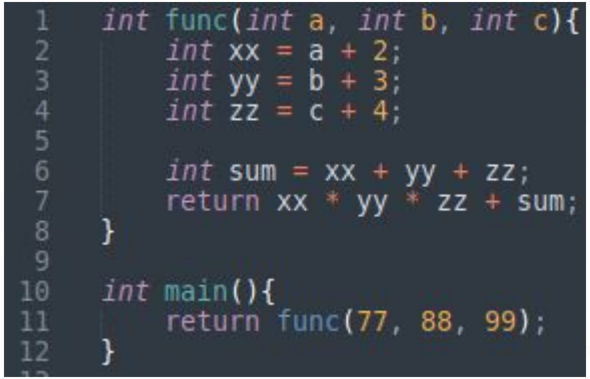
```
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```



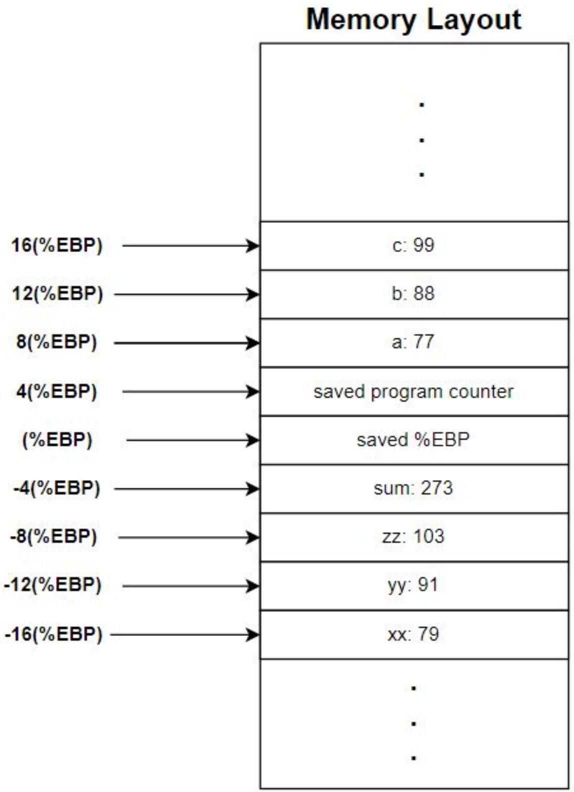
```
func:
.LFB0:
	.cfi_startproc
	endbr32
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	8(%ebp), %eax
	addl	$2, %eax
	movl	%eax, -16(%ebp)
	movl	12(%ebp), %eax
	addl	$3, %eax
	movl	%eax, -12(%ebp)
	movl	16(%ebp), %eax
	addl	$4, %eax
	movl	%eax, -8(%ebp)
	movl	-16(%ebp), %edx
	movl	-12(%ebp), %eax
	addl	%eax, %edx
	movl	-8(%ebp), %eax
	addl	%edx, %eax
	movl	%eax, -4(%ebp)
	movl	-16(%ebp), %eax
	imull	-12(%ebp), %eax
	imull	-8(%ebp), %eax
	movl	%eax, %edx
	movl	-4(%ebp), %eax
	addl	%edx, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

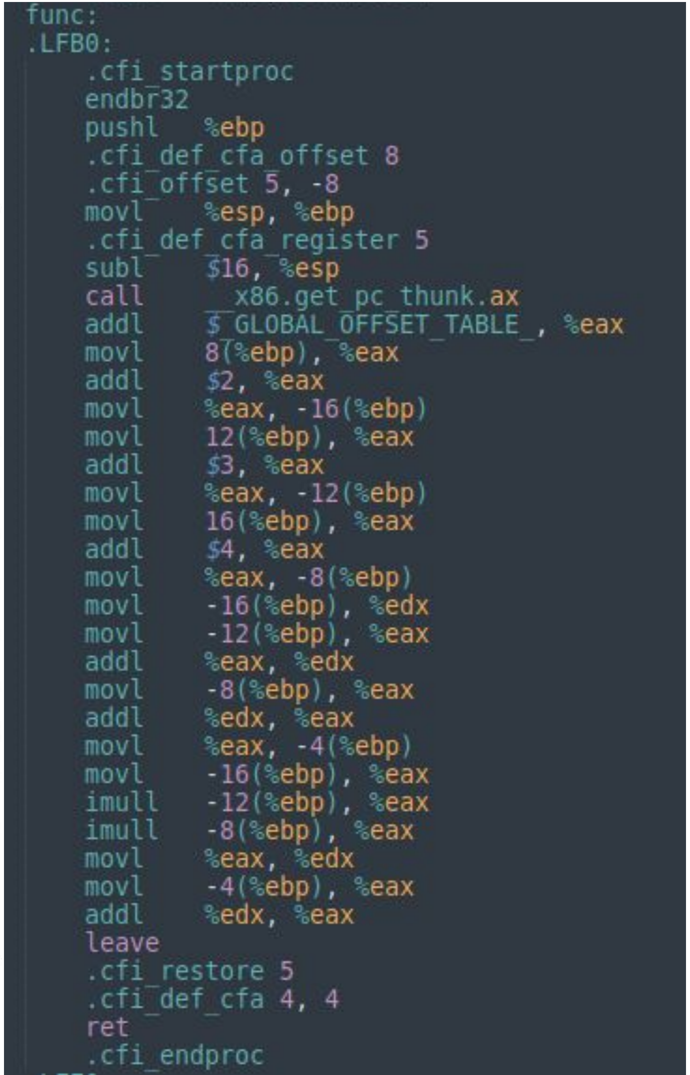
Return value is stored in %EAX

Example 2



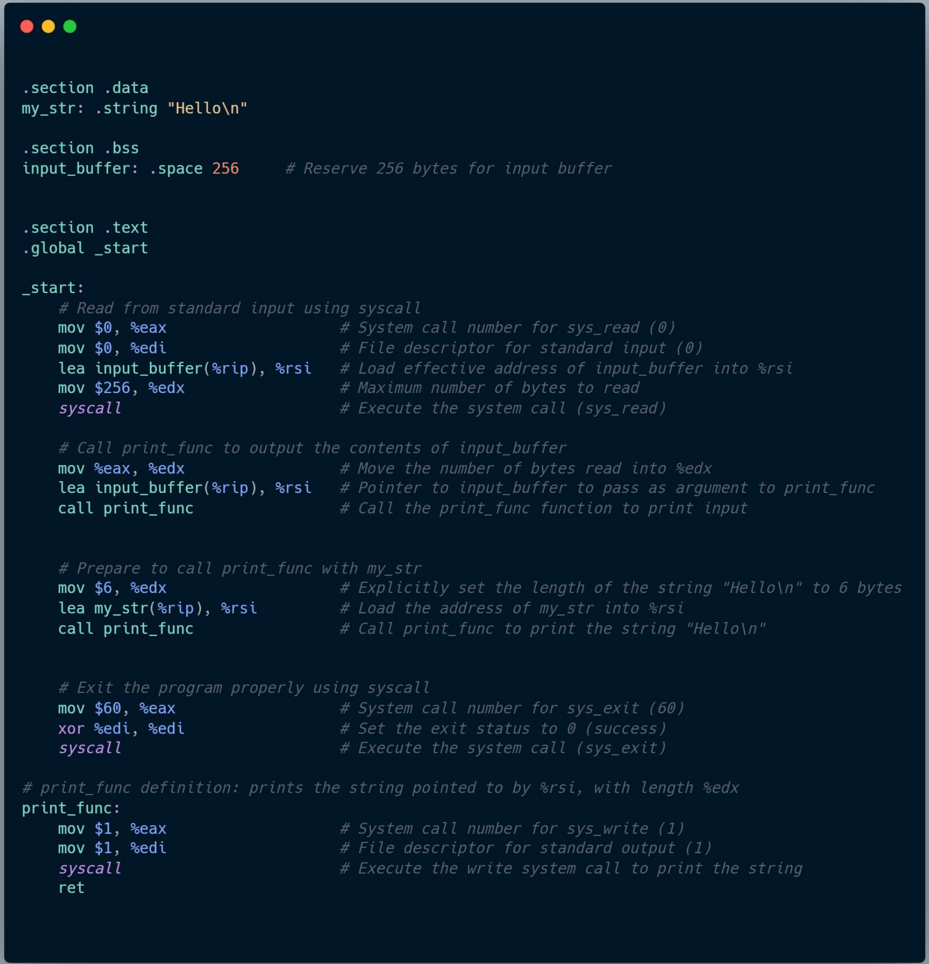
```c
int func(int a, int b, int c){
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;

    int sum = xx + yy + zz;
    return xx * yy * zz + sum;
}

int main(){
    return func(77, 88, 99);
}
```





```
func:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    8(%ebp), %eax
        addl    $2, %eax
        movl    %eax, -16(%ebp)
        movl    12(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -12(%ebp)
        movl    16(%ebp), %eax
        addl    $4, %eax
        movl    %eax, -8(%ebp)
        movl    -16(%ebp), %edx
        movl    -12(%ebp), %eax
        addl    %eax, %edx
        movl    -8(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        movl    -16(%ebp), %eax
        imull   -12(%ebp), %eax
        imull   -8(%ebp), %eax
        movl    %eax, %edx
        movl    -4(%ebp), %eax
        addl    %edx, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

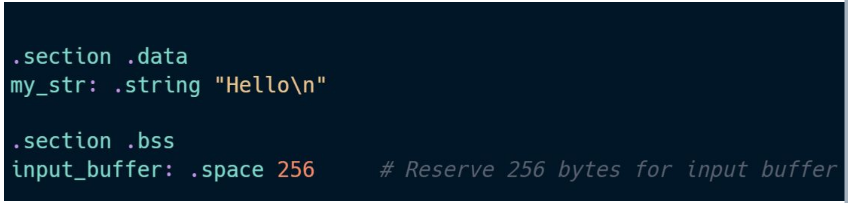
Example 3



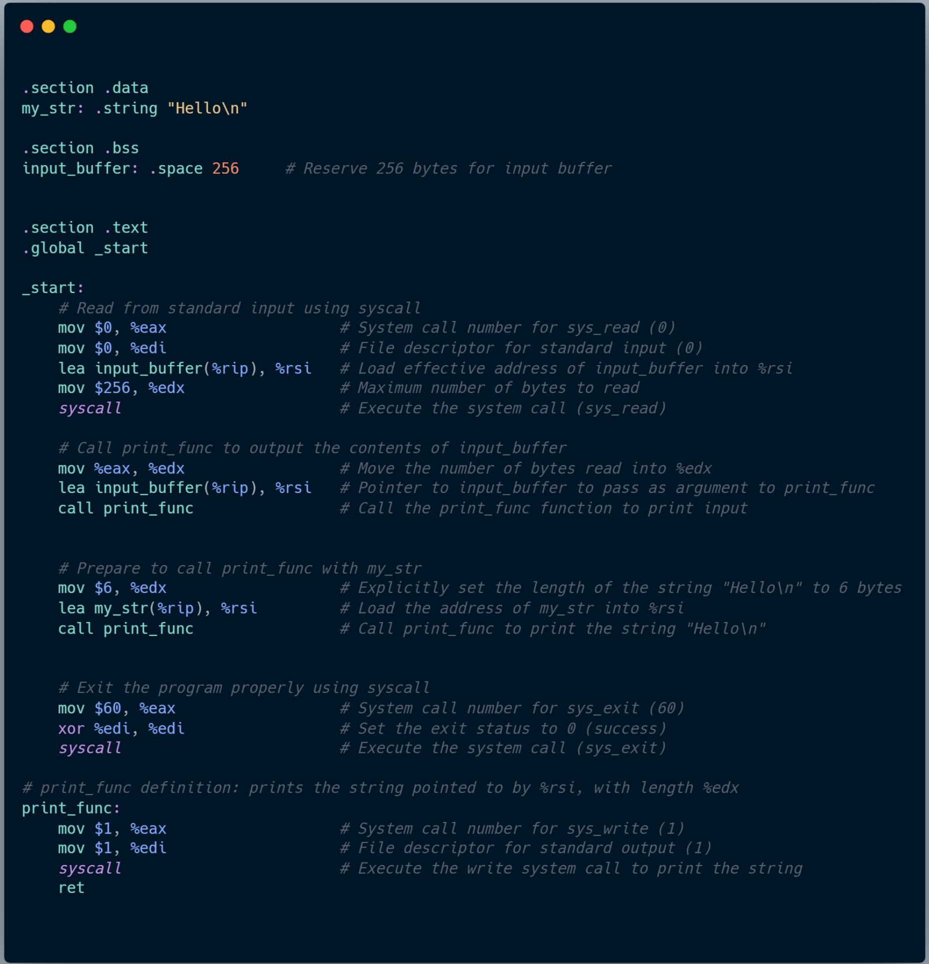
```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start

_start:
    # Read from standard input using syscall
    mov $0, %eax                   # System call number for sys_read (0)
    mov $0, %edi                   # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi   # Load effective address of input_buffer into %rsi
    mov $256, %edx                 # Maximum number of bytes to read
    syscall                        # Execute the system call (sys_read)

    # Call print_func to output the contents of input_buffer
    mov %eax, %edx                 # Move the number of bytes read into %edx
    lea input_buffer(%rip), %rsi   # Pointer to input_buffer to pass as argument to print_func
    call print_func                # Call the print_func function to print input


    # Prepare to call print_func with my_str
    mov $6, %edx                   # Explicitly set the length of the string "Hello\n" to 6 bytes
    lea my_str(%rip), %rsi         # Load the address of my_str into %rsi
    call print_func                # Call print_func to print the string "Hello\n"


    # Exit the program properly using syscall
    mov $60, %eax                  # System call number for sys_exit (60)
    xor %edi, %edi                 # Set the exit status to 0 (success)
    syscall                        # Execute the system call (sys_exit)

# print_func definition: prints the string pointed to by %rsi, with length %edx
print_func:
    mov $1, %eax                   # System call number for sys_write (1)
    mov $1, %edi                   # File descriptor for standard output (1)
    syscall                        # Execute the write system call to print the string
    ret
```

Example 3

Data and BSS



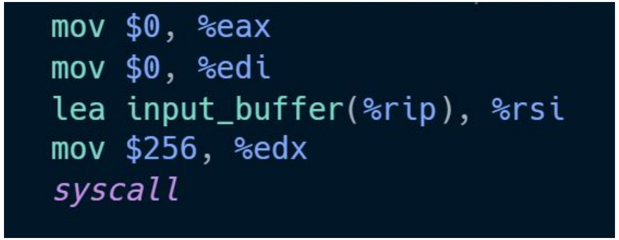
```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer
```



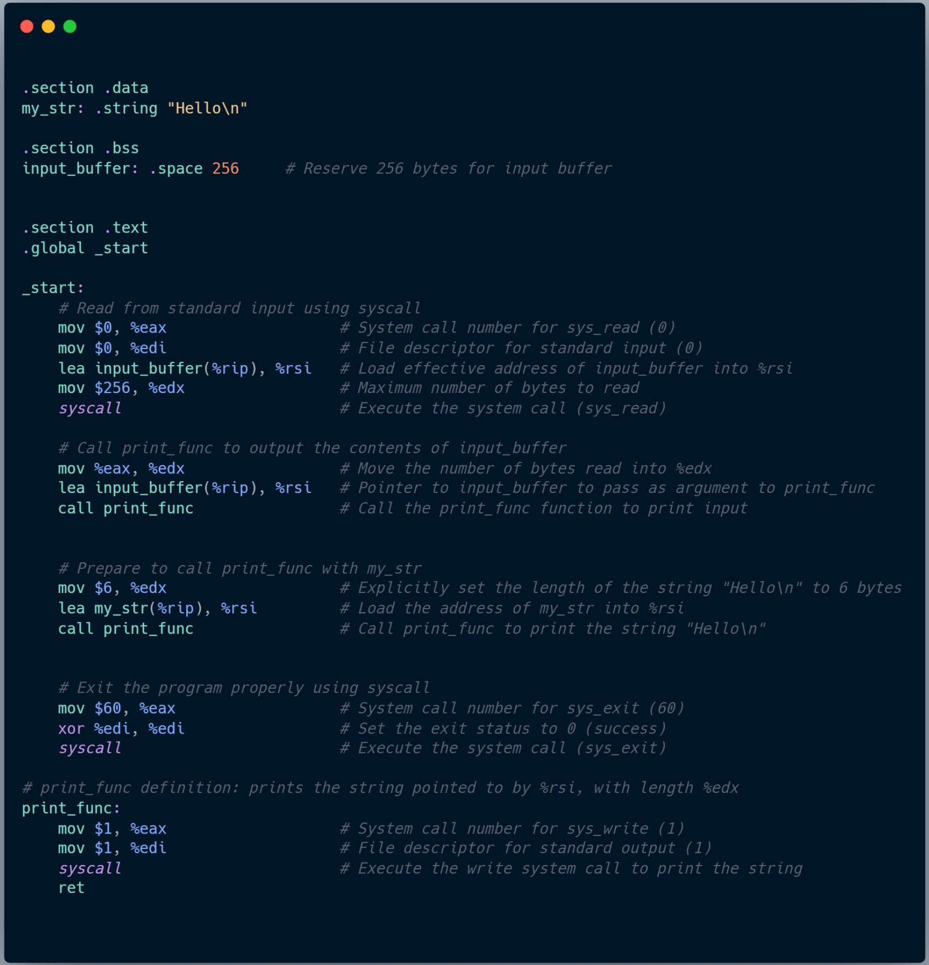
```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start

_start:
    # Read from standard input using syscall
    mov $0, %eax                   # System call number for sys_read (0)
    mov $0, %edi                   # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi   # Load effective address of input_buffer into %rsi
    mov $256, %edx                 # Maximum number of bytes to read
    syscall                        # Execute the system call (sys_read)

    # Call print_func to output the contents of input_buffer
    mov %eax, %edx                 # Move the number of bytes read into %edx
    lea input_buffer(%rip), %rsi   # Pointer to input_buffer to pass as argument to print_func
    call print_func                # Call the print_func function to print input


    # Prepare to call print_func with my_str
    mov $6, %edx                   # Explicitly set the length of the string "Hello\n" to 6 bytes
    lea my_str(%rip), %rsi         # Load the address of my_str into %rsi
    call print_func                # Call print_func to print the string "Hello\n"


    # Exit the program properly using syscall
    mov $60, %eax                  # System call number for sys_exit (60)
    xor %edi, %edi                 # Set the exit status to 0 (success)
    syscall                        # Execute the system call (sys_exit)

# print_func definition: prints the string pointed to by %rsi, with length %edx
print_func:
    mov $1, %eax                   # System call number for sys_write (1)
    mov $1, %edi                   # File descriptor for standard output (1)
    syscall                        # Execute the write system call to print the string
    ret
```

Example 3

Input



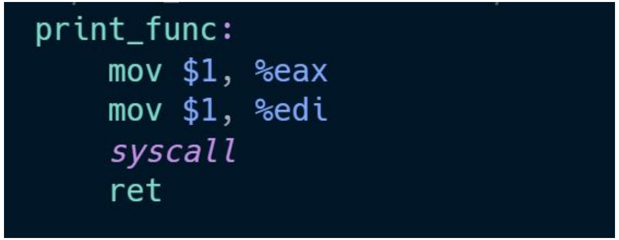
```
mov $0, %eax
mov $0, %edi
lea input_buffer(%rip), %rsi
mov $256, %edx
syscall
```



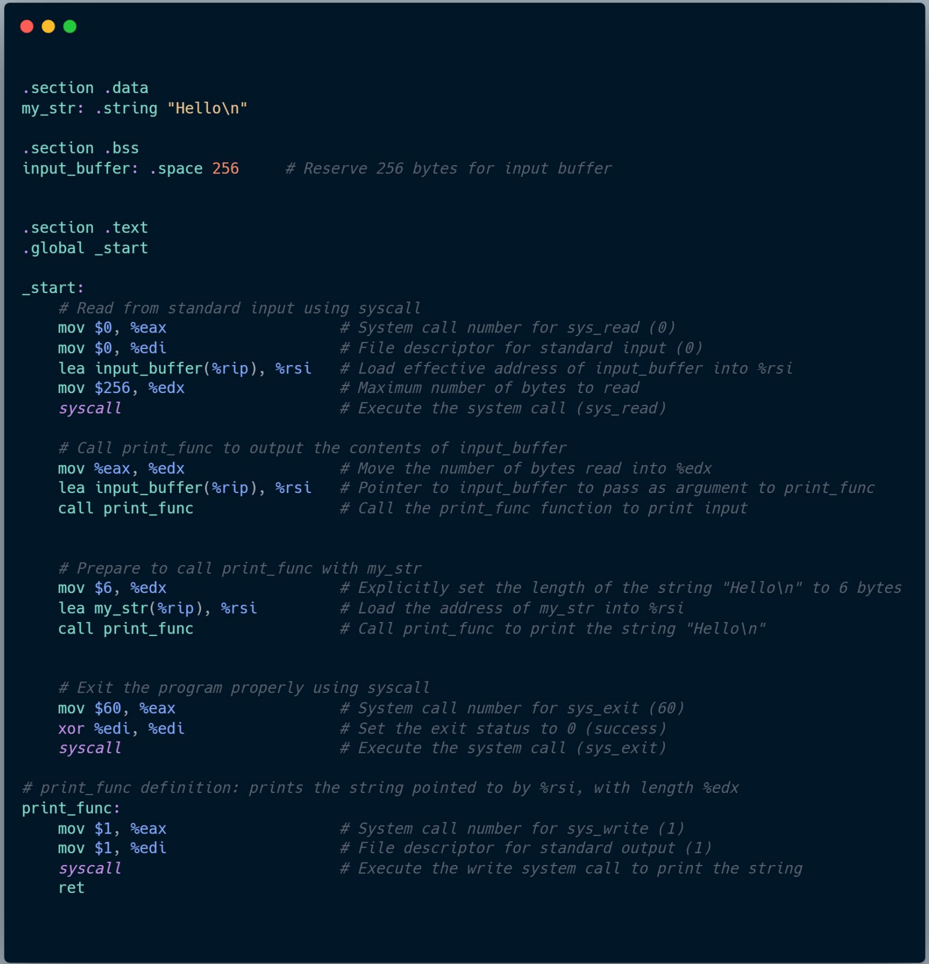
```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start

_start:
    # Read from standard input using syscall
    mov $0, %eax                   # System call number for sys_read (0)
    mov $0, %edi                   # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi   # Load effective address of input_buffer into %rsi
    mov $256, %edx                 # Maximum number of bytes to read
    syscall                        # Execute the system call (sys_read)

    # Call print_func to output the contents of input_buffer
    mov %eax, %edx                 # Move the number of bytes read into %edx
    lea input_buffer(%rip), %rsi   # Pointer to input_buffer to pass as argument to print_func
    call print_func                # Call the print_func function to print input


    # Prepare to call print_func with my_str
    mov $6, %edx                   # Explicitly set the length of the string "Hello\n" to 6 bytes
    lea my_str(%rip), %rsi         # Load the address of my_str into %rsi
    call print_func                # Call print_func to print the string "Hello\n"


    # Exit the program properly using syscall
    mov $60, %eax                  # System call number for sys_exit (60)
    xor %edi, %edi                 # Set the exit status to 0 (success)
    syscall                        # Execute the system call (sys_exit)

# print_func definition: prints the string pointed to by %rsi, with length %edx
print_func:
    mov $1, %eax                   # System call number for sys_write (1)
    mov $1, %edi                   # File descriptor for standard output (1)
    syscall                        # Execute the write system call to print the string
    ret
```

Example 3

Print function



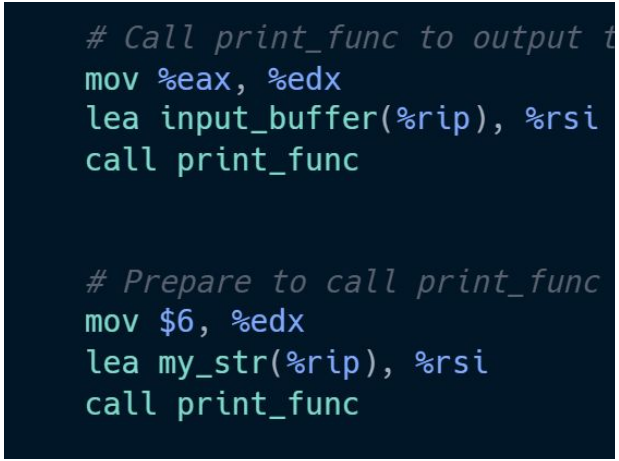
```
print_func:
    mov $1, %eax
    mov $1, %edi
    syscall
    ret
```



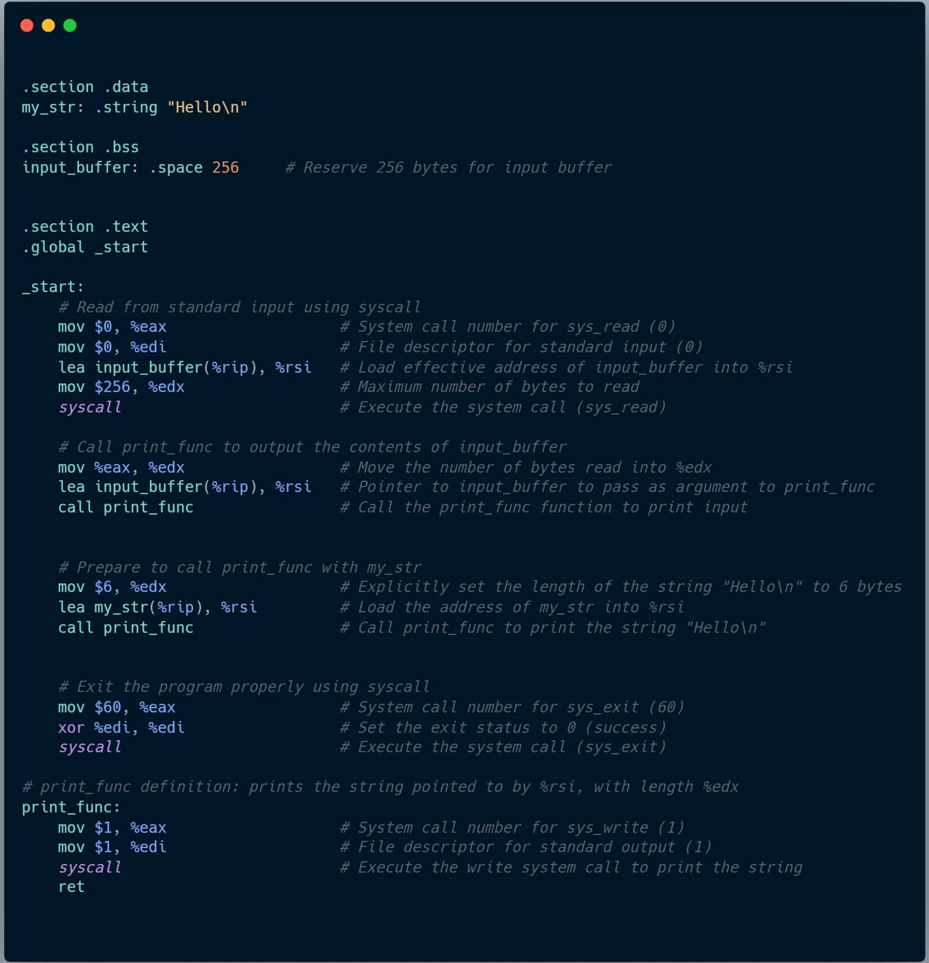
```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start

_start:
    # Read from standard input using syscall
    mov $0, %eax                   # System call number for sys_read (0)
    mov $0, %edi                   # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi   # Load effective address of input_buffer into %rsi
    mov $256, %edx                 # Maximum number of bytes to read
    syscall                        # Execute the system call (sys_read)

    # Call print_func to output the contents of input_buffer
    mov %eax, %edx                 # Move the number of bytes read into %edx
    lea input_buffer(%rip), %rsi   # Pointer to input_buffer to pass as argument to print_func
    call print_func                # Call the print_func function to print input


    # Prepare to call print_func with my_str
    mov $6, %edx                   # Explicitly set the length of the string "Hello\n" to 6 bytes
    lea my_str(%rip), %rsi         # Load the address of my_str into %rsi
    call print_func                # Call print_func to print the string "Hello\n"


    # Exit the program properly using syscall
    mov $60, %eax                  # System call number for sys_exit (60)
    xor %edi, %edi                 # Set the exit status to 0 (success)
    syscall                        # Execute the system call (sys_exit)

# print_func definition: prints the string pointed to by %rsi, with length %edx
print_func:
    mov $1, %eax                   # System call number for sys_write (1)
    mov $1, %edi                   # File descriptor for standard output (1)
    syscall                        # Execute the write system call to print the string
    ret
```

Example 3

Printing data



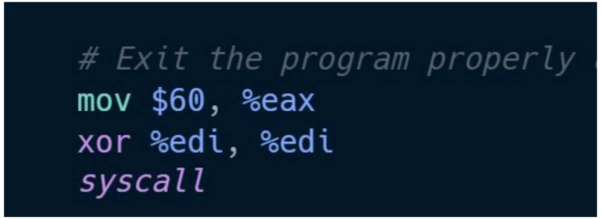
```
# Call print_func to output t
mov %eax, %edx
lea input_buffer(%rip), %rsi
call print_func


# Prepare to call print_func
mov $6, %edx
lea my_str(%rip), %rsi
call print_func
```



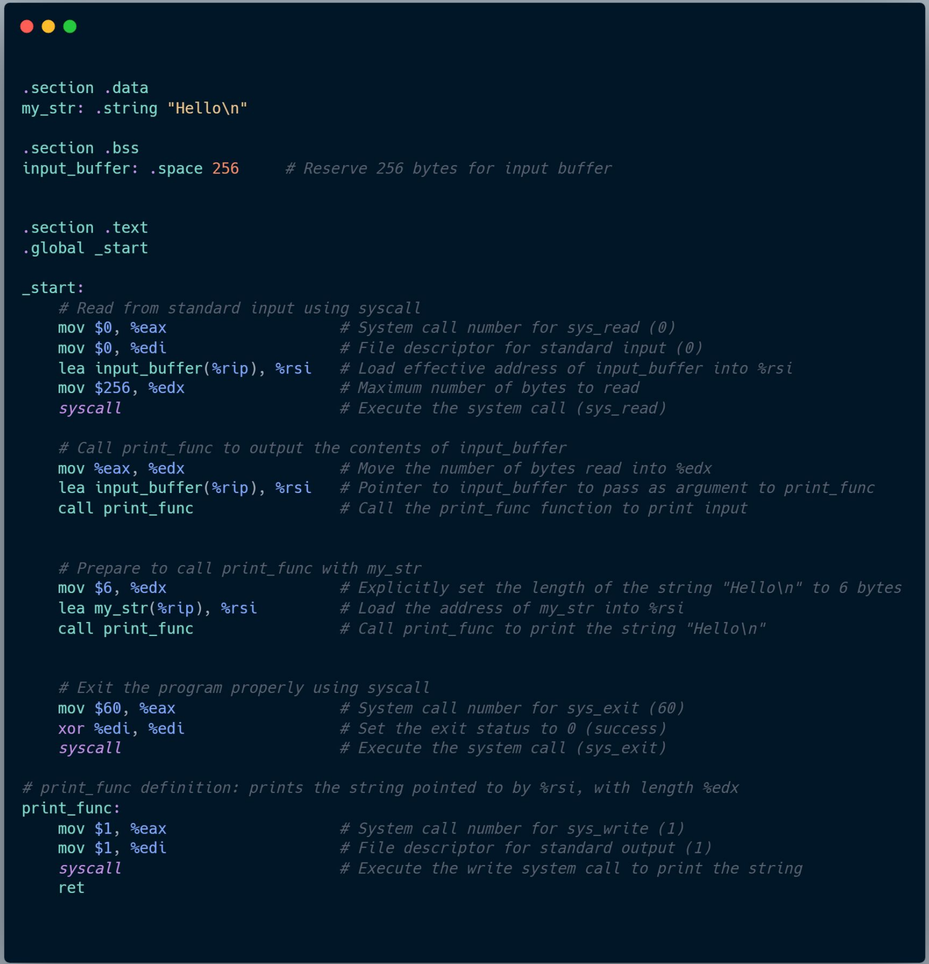
```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start

_start:
    # Read from standard input using syscall
    mov $0, %eax                   # System call number for sys_read (0)
    mov $0, %edi                   # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi   # Load effective address of input_buffer into %rsi
    mov $256, %edx                 # Maximum number of bytes to read
    syscall                        # Execute the system call (sys_read)

    # Call print_func to output the contents of input_buffer
    mov %eax, %edx                 # Move the number of bytes read into %edx
    lea input_buffer(%rip), %rsi   # Pointer to input_buffer to pass as argument to print_func
    call print_func                # Call the print_func function to print input


    # Prepare to call print_func with my_str
    mov $6, %edx                   # Explicitly set the length of the string "Hello\n" to 6 bytes
    lea my_str(%rip), %rsi         # Load the address of my_str into %rsi
    call print_func                # Call print_func to print the string "Hello\n"


    # Exit the program properly using syscall
    mov $60, %eax                  # System call number for sys_exit (60)
    xor %edi, %edi                 # Set the exit status to 0 (success)
    syscall                        # Execute the system call (sys_exit)

# print_func definition: prints the string pointed to by %rsi, with length %edx
print_func:
    mov $1, %eax                   # System call number for sys_write (1)
    mov $1, %edi                   # File descriptor for standard output (1)
    syscall                        # Execute the write system call to print the string
    ret
```

Example 3

Exit routine



```
# Exit the program properly
mov $60, %eax
xor %edi, %edi
syscall
```



```
.section .data
my_str: .string "Hello\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start

_start:
    # Read from standard input using syscall
    mov $0, %eax                   # System call number for sys_read (0)
    mov $0, %edi                   # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi   # Load effective address of input_buffer into %rsi
    mov $256, %edx                 # Maximum number of bytes to read
    syscall                        # Execute the system call (sys_read)

    # Call print_func to output the contents of input_buffer
    mov %eax, %edx                 # Move the number of bytes read into %edx
    lea input_buffer(%rip), %rsi   # Pointer to input_buffer to pass as argument to print_func
    call print_func                # Call the print_func function to print input


    # Prepare to call print_func with my_str
    mov $6, %edx                   # Explicitly set the length of the string "Hello\n" to 6 bytes
    lea my_str(%rip), %rsi         # Load the address of my_str into %rsi
    call print_func                # Call print_func to print the string "Hello\n"


    # Exit the program properly using syscall
    mov $60, %eax                  # System call number for sys_exit (60)
    xor %edi, %edi                 # Set the exit status to 0 (success)
    syscall                        # Execute the system call (sys_exit)

# print_func definition: prints the string pointed to by %rsi, with length %edx
print_func:
    mov $1, %eax                   # System call number for sys_write (1)
    mov $1, %edi                   # File descriptor for standard output (1)
    syscall                        # Execute the write system call to print the string
    ret
```